

C10

		Function Index	Page 1	Thu Jan 03 12:52:58 2008	Function Index	Page 2
RSTPI_ClearRestoreContext	82	(restore_dcpi/plugin.cc)				
RSTPI_DoesAlternateExist	89	(restore_dcpi/plugin.cc)				
RSTPI_Finish	88	(restore_dcpi/plugin.cc)				
RSTPI_GetAllBackupTimes	54	(restore_dcpi/plugin.cc)				
RSTPI_GetCurrentBackupTime	57	(restore_dcpi/plugin.cc)				
RSTPI_GetNextLevelObjects	37	(restore_dcpi/plugin.cc)				
RSTPI_GetTopLevelObjects	29	(restore_dcpi/plugin.cc)				
RSTPI_GetTopLevelTemplates	79	(restore_dcpi/plugin.cc)				
RSTPI_Identify	87	(restore_dcpi/plugin.cc)				
RSTPI_Initialize	24	(restore_dcpi/plugin.cc)				
RSTPI_IsObjectMarkable	50	(restore_dcpi/plugin.cc)				
RSTPI_IsObjectMarkable	78	(restore_dcpi/plugin.cc)				
RSTPI_IsObjectMarked	52	(restore_dcpi/plugin.cc)				
RSTPI_IsThereNextBackup	77	(restore_dcpi/plugin.cc)				
RSTPI_IsTherePrevBackup	85	(restore_dcpi/plugin.cc)				
RSTPI_IsTherePrevBackupForTime	74	(restore_dcpi/plugin.cc)				
RSTPI_IsTherePrevBackupForTime	83	(restore_dcpi/plugin.cc)				
RSTPI_MarkObject	44	(restore_dcpi/plugin.cc)				
RSTPI_SetBackupForTime	59	(restore_dcpi/plugin.cc)				
RSTPI_SetFirstBackup	71	(restore_dcpi/plugin.cc)				
RSTPI_SetMostRecentBackup	68	(restore_dcpi/plugin.cc)				
RSTPI_SetNextBackup	65	(restore_dcpi/plugin.cc)				
RSTPI_SetPrevBackup	62	(restore_dcpi/plugin.cc)				
RSTPI_SetTopLevelObject	33	(restore_dcpi/plugin.cc)				
RSTPI_Submit	27	(restore_dcpi/plugin.cc)				
RSTPI_UnmarkObject	47	(restore_dcpi/plugin.cc)				
RSTSL_Finish	5	(pgms_restore/RStlinitfin.c)				
RSTSL_Initialize	3	(pgms_restore/RStlinitfin.c)				
dump_tlo_list	19	(restore_dcpi/plugin.cc)				
dump_time_list	20	(restore_dcpi/plugin.cc)				
dump_unix_time	18	(restore_dcpi/plugin.cc)				
dump_uro_list	22	(restore_dcpi/plugin.cc)				
init_plugins	9	(pgms_restore/RStlinitfin.c)				
validate_plugin	13	(pgms_restore/RStlinitfin.c)				

pgms_restore/RSLinitfin.c	1
RSPSL_Finish.....	5
RSPSL_Initialize.....	3
init_plugins.....	9
validate_plugin.....	13
restore_dcpilplugin.cc	17
RSTPI_ClearRestoreContext.....	82
RSTPI_DoesAlternateExist.....	89
RSTPI_Finish.....	88
RSTPI_GetAllBackupTimes.....	54
RSTPI_GetCurrentBackupTime.....	57
RSTPI_GetNextLevelObjects.....	37
RSTPI_GetTopLevelObjects.....	29
RSTPI_GetTopLevelTemplates.....	79
RSTPI_Identify.....	87
RSTPI_Initialize.....	24
RSTPI_IsObjectMarkable.....	50
RSTPI_IsObjectMarkable.....	78
RSTPI_IsObjectMarked.....	52
RSTPI_IsThereNextBackup.....	77
RSTPI_IsThereNextBackupForTime.....	85
RSTPI_IsTherePrevBackup.....	74
RSTPI_IsTherePrevBackupForTime.....	83
RSTPI_MarkObject.....	44
RSTPI_SetBackupForTime.....	59
RSTPI_SetFirstBackup.....	71
RSTPI_SetMostRecentBackup.....	68
RSTPI_SetNextBackup.....	65
RSTPI_SetPrevBackup.....	62
RSTPI_SetTopLevelObject.....	33
RSTPI_Submit.....	27
RSTPI_UnmarkObject.....	47
dump_time_list.....	19
dump_tlo_list.....	20
dump_unix_time.....	18
dump_uro_list.....	22


```

2   /*
3   * File Name: RSLinitfin.c
4   */
5   /**
6   * Copyright (c) 1998, 1999 by EMC Corporation.
7   */
8   /**
9   * Purpose:
10  *   This module contains the Restore Service Library
11  *   functions to
12  *   initialize and terminate the restore operation.
13  */
14  /**
15  *   RSLSL_Finish
16  */
17  /**
18  *   Internal Functions:
19  */
20  /**
21  *   Compile-Time Options:
22  *   This section must list any compile time definitions
23  *   which will affect this header.
24  */
25  /**
26  *   The following provides an RCS id in the binary that can be located
27  *   with the what(1) utility. The intent is to keep this short.
28  */
29  /**
30  *ifndef lint
31  *static char RCS_id [] = "$RCSfile$"
32  *           "$Revision$"
33  *           "$Date$";
34  */
35  /**
36  */
37  /**
38  * Feature test switches.
39  */
40  /**
41  * Standard defines required to turn on OS features go here.
42  */
43  /**
44  * The following is required for code that uses POSIX API's.
45  */
46  /**
47  * Remove for non-POSIX, non-portable code.
48  */
49  /**
50  * System headers.
51  */
52  /**
53  *include <sys/param.h>
54  */
55  /**
56  */
57  /**
58  * Epoch headers.
59  */
60  /**
61  */
62  /**
63  */
64  /**
65  * Local headers
66  */
67  #include <RSLinterns.h>

```

```

68
69  /**
70  * #defines, structures, typedefs local to this source file
71  */
72  /**
73  * static errno_ty init_plugins( restore_context *rcp );
74  * static int validate_plugin( struct pluginData *pDataPtr );
75  */
76
77  /**
78  * External declarations
79  */
80
81  /**
82  * This is the global "restore context" that will be used
83  * throughout the rest of the restore operations.
84  */
85  struct restore_context *rcp = NULL;
86
87  /**
88  * Definitions of the names of the plugin functions in the piFuncArray
89  * of the pluginData structure. These must be in the same order and position
90  * as the piFuncIndex values defined in RSLplugin.h.
91  */
92
93  char *piFuncNames[PIFuncIndexLast+1] = {
94  "RSTPI_Identify",
95  "RSTPI_Initialize",
96  "RSTPI_SetTopLevelObjects",
97  "RSTPI_SetNextLevelObject",
98  "RSTPI_GetNextLevelObjects",
99  "RSTPI_ClearRestoreContext",
100 "RSTPI_Submit",
101 "RSTPI_GetTopLevelTemplate",
102 "RSTPI_DoesAlternateExist",
103 "RSTPI_MarkObject",
104 "RSTPI_UnmarkObject",
105 "RSTPI_IsObjectMarked",
106 "RSTPI_IsObjectMarkable",
107 "RSTPI_SetAllBackupTimes",
108 "RSTPI_GetCurrentBackupTime",
109 "RSTPI_SetBackupForTime",
110 "RSTPI_SetPrevBackup",
111 "RSTPI_SetNextBackup",
112 "RSTPI_SetFirstBackup",
113 "RSTPI_SetMostRecentBackup",
114 "RSTPI_SetPrevBackup",
115 "RSTPI_IsTherePrevBackup",
116 "RSTPI_IsThereNextBackup",
117 "RSTPI_IsThereNextBackupForTime",
118 "RSTPI_IsTherePrevBackupForTime",
119 "RSTPI_StartRestore",
120 "RSTPI_FindRestoreableObjects",
121 "RSTPI_FindNecessaryMedia",
122 "RSTPI_GetFindResults",
123 };

```

```

127
128 * RSMSL_Initialize;
129 *
130 * This function takes care of all the initialization for a restore
131 * session. This must be called prior to any of the other functions
132 * in the Restore API.
133 *
134 * Parameters:
135 *
136 *   * userName (I) - The name of the user.
137 *
138 * ****
139 */
140 RSMSL_Initialize( const char *userName )
141 {
142     errno_ty
143     {
144         errno_ty status = E_SUCCESS;
145
146         /*
147         * If we have not yet allocated space for a restore_context
148         * structure, do so now. If we have already done so, just clear it
149         * now.
150         */
151
152         if (NULL == rcp)
153         {
154             if (NULL == (rcp = (struct restore_context *)malloc(sizeof(
155                 rec_api_log_csm(SUB_CSM_NOMEM, NULL));
156                 return(EP_RB_RECOVER_NOMEM);
157             }
158         }
159
160         memset(rcp, 0, sizeof(struct restore_context));
161
162         rcp->rc_human_uidname = esl_strdup( userName );
163
164         if (!rcp->rc_human_uidname)
165             rec_api_log_csm(SUB_CSM_NOMEM, NULL);
166         return(EP_RB_RECOVER_NOMEM);
167     }
168
169     /*
170     * Set the appropriate field in the recovery context to indicate
171     * that this recover session is based on the Recover API.
172     * This flag is in place for historical reasons but is used by
173     * other parts of the Recover API library.
174     */
175
176     rcb->gui_mode = 1;
177
178     /*
179     * Initialize the logging mechanism.
180     */
181
182     if (status = rbrlog_begin(rcp, programe))
183     {
184         /*
185         * Initialize the few "recover context" variables that we can at
186         */
187
188     /*
189     * Initialize the few "recover context" variables that we can at
190     */

```

```

214
215 	* RSTSL_Finish
216 	*
217 	* Function Description:
218 	*
219 	* This function terminates a restoral session, but not while a restore is in
220 	* progress.
221 	* It will be rejected if a restore is currently being executed.
222 	*
223 	* Parameters:
224 	* none
225 	*
226 	*/
227

228 	errno_ty
229
230 	RSTSL_Finish( void )
231 	{
232 	int mc_n;
233
234 	errno_ty err = E_SUCCESS;
235
236 	if (NULL == rcp)
237 	{
238 		return( E_SUCCESS );
239 	}
240
241 	RemoveSubmitFiles();
242
243 	/* Call rbr_cleanup() which kills the aux proc( s ), unlocks the work
244 	* item, then calls rbrlog_end( ) to enter the last logs and to close
245 	* the log file.
246 	*/
247
248 	rbr_cleanup(rcp);
249
250 	/*
251 	* Deallocate the memory of restore_context and the related
252 	* structures.
253 	*/
254
255 	if (NULL != rcp->rc_mcP) /* Free the multicast structures */
256 	{
257 	ncat_destroy(rcp->rc_mcP);
258 	}
259
260 	/*
261 	* Free the mark bit map space
262 	*/
263
264 	for (mc_n = 0; mc_n < rcp->rc_marks_plane_alloc; mc_n++)
265 	{
266 	if (NULL != rcp->rc_marks[mc_n])
267 	{
268 	free(rcp->rc_marks[mc_n]);
269 	}
270
271 	}
272
273 	if (NULL != rcp->rc_marks_by_plane)
274
275 	RSTSL_Finish
276 	*
277 	* Function Description:
278 	*
279 	* This routine will clean up any local memory used in the session.
280 	*/
281
282 	#if 0
283 	if (NULL != rcp->rc_cfgname)
284 	{
285 	free(rcp->rc_cfgname);
286 	}
287 	#endif
288
289 	if (NULL != rcp->rc_config)
290 	{
291 	rbc_freeconfig(rcp->rc_config);
292 	}
293
294 	/*
295 	* Free the DS_NONE structures array
296 	* Note that even though rc_dsnones is the head of linked list
297 	* of dsnone_info structures, the list is allocated via malloc
298 	* as an array initially (ref. alloc_plane_arrays()), therefore
299 	* we can do a free here.
300 	*/
301
302 	if (NULL != rcp->rc_dsnones)
303 	{
304 	free(rcp->rc_dsnones);
305 	}
306
307 	/*
308 	* Free the volume list structures.
309 	*/
310
311 	if (NULL != rcp->ebvlist)
312 	{
313 	(void)ebvl_voidlist_destructor(
314 	rcp->ebvlist, EBVL_DESTROY_ALL);
315
316 	/*
317 	* Free the plugin related data
318 	*/
319
320 	while (rcp->rc_backup_app = 0;
321 	while (rcp->currentPiptr = rcp->pilist)
322 	{
323 	rcp->rc_backup_app++;
324 	rcp->appData = rcp->currentPiptr->appData;
325 	/* allow plugin to clean up and close .so: */
326 	if (E_SUCCESS != (err =
327 	rcp-> currentPiptr-> piFuncArray[ PIFuncIndexFinish ] ( rcp ) ) )
328 	{
329 	/* log error, continue */
330 	rbe_user_error( err,
331 	"RSTPL_Finish failed for restore plug-in
332 	library %s\n",
333 	( struct pluginIData * (
334 	rcp-> currentPiptr-> idData ) -> name );
335 	}
336 	}
337 	}
338
339 	/* log error, continue */
340 	rbe_user_error( err,
341 	"RSTPL_Finish failed for restore plug-in
342 	library %s\n",
343 	( struct pluginIData * (
344 	rcp-> currentPiptr-> idData ) -> name );
345 	}
346
347 	}
348
349 	/* log error, continue */
350 	rbe_user_error( err,
351 	"RSTPL_Finish failed for restore plug-in
352 	library %s\n",
353 	( struct pluginIData * (
354 	rcp-> currentPiptr-> idData ) -> name );
355 	}
356
357 	}
358
359 	*/
360
361 	/*
362 	* Free the configuration structures
363 	*/
364
365 	/*
366 	* Free the configuration structures
367 	*/
368
369 	/*
370 	* Free the configuration structures
371 	*/
372
373 	/*
374 	* Free the configuration structures
375 	*/
376
377 	/*
378 	* Free the configuration structures
379 	*/
380
381 	/*
382 	* Free the configuration structures
383 	*/
384
385 	/*
386 	* Free the configuration structures
387 	*/
388
389 	/*
390 	* Free the configuration structures
391 	*/
392
393 	/*
394 	* Free the configuration structures
395 	*/
396
397 	/*
398 	* Free the configuration structures
399 	*/
400
401 	/*
402 	* Free the configuration structures
403 	*/
404
405 	/*
406 	* Free the configuration structures
407 	*/
408
409 	/*
410 	* Free the configuration structures
411 	*/
412
413 	/*
414 	* Free the configuration structures
415 	*/
416
417 	/*
418 	* Free the configuration structures
419 	*/
420
421 	/*
422 	* Free the configuration structures
423 	*/
424
425 	/*
426 	* Free the configuration structures
427 	*/
428
429 	/*
430 	* Free the configuration structures
431 	*/
432
433 	/*
434 	* Free the configuration structures
435 	*/
436
437 	/*
438 	* Free the configuration structures
439 	*/
440
441 	/*
442 	* Free the configuration structures
443 	*/
444
445 	/*
446 	* Free the configuration structures
447 	*/
448
449 	/*
450 	* Free the configuration structures
451 	*/
452
453 	/*
454 	* Free the configuration structures
455 	*/
456
457 	/*
458 	* Free the configuration structures
459 	*/
460
461 	/*
462 	* Free the configuration structures
463 	*/
464
465 	/*
466 	* Free the configuration structures
467 	*/
468
469 	/*
470 	* Free the configuration structures
471 	*/
472
473 	/*
474 	* Free the configuration structures
475 	*/
476
477 	/*
478 	* Free the configuration structures
479 	*/
480
481 	/*
482 	* Free the configuration structures
483 	*/
484
485 	/*
486 	* Free the configuration structures
487 	*/
488
489 	/*
490 	* Free the configuration structures
491 	*/
492
493 	/*
494 	* Free the configuration structures
495 	*/
496
497 	/*
498 	* Free the configuration structures
499 	*/
499
500 	/*
501 	* Free the configuration structures
502 	*/
503
504 	/*
505 	* Free the configuration structures
506 	*/
507
508 	/*
509 	* Free the configuration structures
510 	*/
511
512 	/*
513 	* Free the configuration structures
514 	*/
515
516 	/*
517 	* Free the configuration structures
518 	*/
519
520 	/*
521 	* Free the configuration structures
522 	*/
523
524 	/*
525 	* Free the configuration structures
526 	*/
527
528 	/*
529 	* Free the configuration structures
530 	*/
531
532 	/*
533 	* Free the configuration structures
534 	*/
535
536 	/*
537 	* Free the configuration structures
538 	*/
539
540 	/*
541 	* Free the configuration structures
542 	*/
543
544 	/*
545 	* Free the configuration structures
546 	*/
547
548 	/*
549 	* Free the configuration structures
550 	*/
551
552 	/*
553 	* Free the configuration structures
554 	*/
555
556 	/*
557 	* Free the configuration structures
558 	*/
559
560 	/*
561 	* Free the configuration structures
562 	*/
563
564 	/*
565 	* Free the configuration structures
566 	*/
567
568 	/*
569 	* Free the configuration structures
570 	*/
571
572 	/*
573 	* Free the configuration structures
574 	*/
575
576 	/*
577 	* Free the configuration structures
578 	*/
579
579
580 	/*
581 	* Free the configuration structures
582 	*/
583
584 	/*
585 	* Free the configuration structures
586 	*/
587
588 	/*
589 	* Free the configuration structures
590 	*/
591
592 	/*
593 	* Free the configuration structures
594 	*/
595
596 	/*
597 	* Free the configuration structures
598 	*/
599
599
600 	/*
601 	* Free the configuration structures
602 	*/
603
604 	/*
605 	* Free the configuration structures
606 	*/
607
608 	/*
609 	* Free the configuration structures
610 	*/
611
612 	/*
613 	* Free the configuration structures
614 	*/
615
616 	/*
617 	* Free the configuration structures
618 	*/
619
619
620 	/*
621 	* Free the configuration structures
622 	*/
623
624 	/*
625 	* Free the configuration structures
626 	*/
627
628 	/*
629 	* Free the configuration structures
630 	*/
631
632 	/*
633 	* Free the configuration structures
634 	*/
635
635
636 	/*
637 	* Free the configuration structures
638 	*/
639
640 	/*
641 	* Free the configuration structures
642 	*/
643
644 	/*
645 	* Free the configuration structures
646 	*/
647
648 	/*
649 	* Free the configuration structures
650 	*/
651
652 	/*
653 	* Free the configuration structures
654 	*/
655
656 	/*
657 	* Free the configuration structures
658 	*/
659
659
660 	/*
661 	* Free the configuration structures
662 	*/
663
664 	/*
665 	* Free the configuration structures
666 	*/
667
668 	/*
669 	* Free the configuration structures
670 	*/
671
672 	/*
673 	* Free the configuration structures
674 	*/
675
676 	/*
677 	* Free the configuration structures
678 	*/
679
679
680 	/*
681 	* Free the configuration structures
682 	*/
683
684 	/*
685 	* Free the configuration structures
686 	*/
687
688 	/*
689 	* Free the configuration structures
690 	*/
691
692 	/*
693 	* Free the configuration structures
694 	*/
695
695
696 	/*
697 	* Free the configuration structures
698 	*/
699
700 	/*
701 	* Free the configuration structures
702 	*/
703
704 	/*
705 	* Free the configuration structures
706 	*/
707
708 	/*
709 	* Free the configuration structures
710 	*/
711
712 	/*
713 	* Free the configuration structures
714 	*/
715
715
716 	/*
717 	* Free the configuration structures
718 	*/
719
720 	/*
721 	* Free the configuration structures
722 	*/
723
724
725 	/*
726 	* Free the configuration structures
727 	*/
728
729 	/*
730 	* Free the configuration structures
731 	*/
732
733 	/*
734 	* Free the configuration structures
735 	*/
735
736
737 	/*
738 	* Free the configuration structures
739 	*/
740
741 	/*
742 	* Free the configuration structures
743 	*/
744
745
746 	/*
747 	* Free the configuration structures
748 	*/
749
750 	/*
751 	* Free the configuration structures
752 	*/
753
754 	/*
755 	* Free the configuration structures
756 	*/
757
758 	/*
759 	* Free the configuration structures
760 	*/
761
762 	/*
763 	* Free the configuration structures
764 	*/
765
766 	/*
767 	* Free the configuration structures
768 	*/
769
770 	/*
771 	* Free the configuration structures
772 	*/
773
774
775 	/*
776 	* Free the configuration structures
777 	*/
778
779 	/*
780 	* Free the configuration structures
781 	*/
782
783 	/*
784 	* Free the configuration structures
785 	*/
786
787 	/*
788 	* Free the configuration structures
789 	*/
789
790
791 	/*
792 	* Free the configuration structures
793 	*/
794
795 	/*
796 	* Free the configuration structures
797 	*/
798
799 	/*
799
800 	* Free the configuration structures
801 	*/
802
803 	/*
804 	* Free the configuration structures
805 	*/
806
807 	/*
808 	* Free the configuration structures
809 	*/
810
811 	/*
812 	* Free the configuration structures
813 	*/
814
814
815 	/*
816 	* Free the configuration structures
817 	*/
818
819 	/*
819
820 	* Free the configuration structures
821 	*/
822
823 	/*
824 	* Free the configuration structures
825 	*/
826
827 	/*
828 	* Free the configuration structures
829 	*/
830
831 	/*
832 	* Free the configuration structures
833 	*/
834
835 	/*
836 	* Free the configuration structures
837 	*/
838
839 	/*
840 	* Free the configuration structures
841 	*/
842
843 	/*
844 	* Free the configuration structures
845 	*/
846
847 	/*
848 	* Free the configuration structures
849 	*/
850
851 	/*
852 	* Free the configuration structures
853 	*/
854
855 	/*
856 	* Free the configuration structures
857 	*/
858
859 	/*
859
860 	* Free the configuration structures
861 	*/
862
863 	/*
864 	* Free the configuration structures
865 	*/
866
867 	/*
868 	* Free the configuration structures
869 	*/
870
871 	/*
872 	* Free the configuration structures
873 	*/
874
875 	/*
876 	* Free the configuration structures
877 	*/
878
879 	/*
880 	* Free the configuration structures
881 	*/
882
883 	/*
884 	* Free the configuration structures
885 	*/
886
887 	/*
888 	* Free the configuration structures
889 	*/
889
890
891 	/*
892 	* Free the configuration structures
893 	*/
894
895 	/*
896 	* Free the configuration structures
897 	*/
898
899 	/*
899
900 	* Free the configuration structures
901 	*/
902
903 	/*
904 	* Free the configuration structures
905 	*/
906
907 	/*
908 	* Free the configuration structures
909 	*/
910
911 	/*
912 	* Free the configuration structures
913 	*/
914
914
915 	/*
916 	* Free the configuration structures
917 	*/
918
919 	/*
919
920 	* Free the configuration structures
921 	*/
922
923 	/*
924 	* Free the configuration structures
925 	*/
926
927 	/*
928 	* Free the configuration structures
929 	*/
930
931 	/*
932 	* Free the configuration structures
933 	*/
934
935 	/*
936 	* Free the configuration structures
937 	*/
938
939 	/*
940 	* Free the configuration structures
941 	*/
942
943 	/*
944 	* Free the configuration structures
945 	*/
946
947 	/*
948 	* Free the configuration structures
949 	*/
950
951 	/*
952 	* Free the configuration structures
953 	*/
954
955 	/*
956 	* Free the configuration structures
957 	*/
958
959 	/*
959
960 	* Free the configuration structures
961 	*/
962
963 	/*
964 	* Free the configuration structures
965 	*/
966
967 	/*
968 	* Free the configuration structures
969 	*/
970
971 	/*
972 	* Free the configuration structures
973 	*/
974
975 	/*
976 	* Free the configuration structures
977 	*/
978
979 	/*
980 	* Free the configuration structures
981 	*/
982
983 	/*
984 	* Free the configuration structures
985 	*/
986
987 	/*
988 	* Free the configuration structures
989 	*/
989
990
991 	/*
992 	* Free the configuration structures
993 	*/
994
995 	/*
996 	* Free the configuration structures
997 	*/
998
999 	/*
999
1000 	* Free the configuration structures
1001 	*/
1002
1003 	/*
1004 	* Free the configuration structures
1005 	*/
1006
1007 	/*
1008 	* Free the configuration structures
1009 	*/
1010
1011 	/*
1012 	* Free the configuration structures
1013 	*/
1014
1014
1015 	/*
1016 	* Free the configuration structures
1017 	*/
1018
1019 	/*
1019
1020 	* Free the configuration structures
1021 	*/
1022
1023 	/*
1024 	* Free the configuration structures
1025 	*/
1026
1027 	/*
1028 	* Free the configuration structures
1029 	*/
1030
1031 	/*
1032 	* Free the configuration structures
1033 	*/
1034
1035 	/*
1036 	* Free the configuration structures
1037 	*/
1038
1039 	/*
1040 	* Free the configuration structures
1041 	*/
1042
1043 	/*
1044 	* Free the configuration structures
1045 	*/
1046
1047 	/*
1048 	* Free the configuration structures
1049 	*/
1049
1050
1051 	/*
1052 	* Free the configuration structures
1053 	*/
1054
1055 	/*
1056 	* Free the configuration structures
1057 	*/
1058
1059 	/*
1060 	* Free the configuration structures
1061 	*/
1062
1063 	/*
1064 	* Free the configuration structures
1065 	*/
1066
1067 	/*
1068 	* Free the configuration structures
1069 	*/
1070
1071 	/*
1072 	* Free the configuration structures
1073 	*/
1074
1075 	/*
1076 	* Free the configuration structures
1077 	*/
1078
1079 	/*
1080 	* Free the configuration structures
1081 	*/
1082
1083 	/*
1084 	* Free the configuration structures
1085 	*/
1086
1087 	/*
1088 	* Free the configuration structures
1089 	*/
1089
1090
1091 	/*
1092 	* Free the configuration structures
1093 	*/
1094
1095 	/*
1096 	* Free the configuration structures
1097 	*/
1098
1099 	/*
1100 	* Free the configuration structures
1101 	*/
1102
1103 	/*
1104 	* Free the configuration structures
1105 	*/
1106
1107 	/*
1108 	* Free the configuration structures
1109 	*/
1110
1111 	/*
1112 	* Free the configuration structures
1113 	*/
1114
1115 	/*
1116 	* Free the configuration structures
1117 	*/
1118
1119 	/*
1120 	* Free the configuration structures
1121 	*/
1122
1123 	/*
1124 	* Free the configuration structures
1125 	*/
1126
1127 	/*
1128 	* Free the configuration structures
1129 	*/
1130
1131 	/*
1132 	* Free the configuration structures
1133 	*/
1134
1135 	/*
1136 	* Free the configuration structures
1137 	*/
1138
1139 	/*
1140 	* Free the configuration structures
1141 	*/
1142
1143 	/*
1144 	* Free the configuration structures
1145 	*/
1146
1147 	/*
1148 	* Free the configuration structures
1149 	*/
1149
1150
1151 	/*
1152 	* Free the configuration structures
1153 	*/
1154
1155 	/*
1156 	* Free the configuration structures
1157 	*/
1158
1159 	/*
1160 	* Free the configuration structures
1161 	*/
1162
1163 	/*
1164 	* Free the configuration structures
1165 	*/
1166
1167 	/*
1168 	* Free the configuration structures
1169 	*/
1169
1170
1171 	/*
1172 	* Free the configuration structures
1173 	*/
1174
1175 	/*
1176 	* Free the configuration structures
1177 	*/
1178
1179 	/*
1180 	* Free the configuration structures
1181 	*/
1182
1183 	/*
1184 	* Free the configuration structures
1185 	*/
1186
1187 	/*
1188 	* Free the configuration structures
1189 	*/
1189
1190
1191 	/*
1192 	* Free the configuration structures
1193 	*/
1194
1195 	/*
1196 	* Free the configuration structures
1197 	*/
1198
1199 	/*
1200 	* Free the configuration structures
1201 	*/
1202
1203 	/*
1204 	* Free the configuration structures
1205 	*/
1206
1207 	/*
1208 	* Free the configuration structures
1209 	*/
1210
1211 	/*
1212 	* Free the configuration structures
1213 	*/
1214
1215 	/*
1216 	* Free the configuration structures
1217 	*/
1218
1219 	/*
1220 	* Free the configuration structures
1221 	*/
1222
1223 	/*
1224 	* Free the configuration structures
1225 	*/
1226
1227 	/*
1228 	* Free the configuration structures
1229 	*/
1229
1230 	/*
1231 	* Free the configuration structures
1232 	*/
1232
1233 	/*
1234 	* Free the configuration structures
1235 	*/
1235
1236 	/*
1237 	* Free the configuration structures
1238 	*/
1238
1239 	/*
1240 	* Free the configuration structures
1241 	*/
1241
1242 	/*
1243 	* Free the configuration structures
1244 	*/
1244
1245 	/*
1246 	* Free the configuration structures
1247 	*/
1247
1248 	/*
1249 	* Free the configuration structures
1250 	*/
1250
1251 	/*
1252 	* Free the configuration structures
1253 	*/
1253
1254 	/*
1255 	* Free the configuration structures
1256 	*/
1256
1257 	/*
1258 	* Free the configuration structures
1259 	*/
1259
1260 	/*
1261 	* Free the configuration structures
1262 	*/
1262
1263 	/*
1264 	* Free the configuration structures
1265 	*/
1265
1266 	/*
1267 	* Free the configuration structures
1268 	*/
1268
1269 	/*
1270 	* Free the configuration structures
1271 	*/
1271
1272 	/*
1273 	* Free the configuration structures
1274 	*/
1274
1275 	/*
1276 	* Free the configuration structures
1277 	*/
1277
1278 	/*
1279 	* Free the configuration structures
1280 	*/
1280
1281 	/*
1282 	* Free the configuration structures
1283 	*/
1283
1284 	/*
1285 	* Free the configuration structures
1286 	*/
1286
1287 	/*
1288 	* Free the configuration structures
1289 	*/
1289
1290 	/*
1291 	* Free the configuration structures
1292 	*/
1292
1293 	/*
1294 	* Free the configuration structures
1295 	*/
1295
1296 	/*
1297 	* Free the configuration structures
1298 	*/
1298
1299 	/*
1300 	* Free the configuration structures
1301 	*/
1301
1302 	/*
1303 	* Free the configuration structures
1304 	*/
1304
1305 	/*
1306 	* Free the configuration structures
1307 	*/
1307
1308 	/*
1309 	* Free the configuration structures
1310 	*/
1310
1311 	/*
1312 	* Free the configuration structures
1313 	*/
1313
1314 	/*
1315 	* Free the configuration structures
1316 	*/
1316
1317 	/*
1318 	* Free the configuration structures
1319 	*/
1319
1320 	/*
1321 	* Free the configuration structures
1322 	*/
1322
1323 	/*
1324 	* Free the configuration structures
1325 	*/
1325
1326 	/*
1327 	* Free the configuration structures
1328 	*/
1328
1329 	/*
1330 	* Free the configuration structures
1331 	*/
1331
1332 	/*
1333 	* Free the configuration structures
1334 	*/
1334
1335 	/*
1336 	* Free the configuration structures
1337 	*/
1337
1338 	/*
1339 	* Free the configuration structures
1340 	*/
1340
1341 	/*
1342 	* Free the configuration structures
1343 	*/
1343
1344 	/*
1345 	* Free the configuration structures
1346 	*/
1346
1347 	/*
1348 	* Free the configuration structures
1349 	*/
1349
1350 	/*
1351 	* Free the configuration structures
1352 	*/
1352
1353 	/*
1354 	* Free the configuration structures
1355 	*/
1355
1356 	/*
1357 	* Free the configuration structures
1358 	*/
1358
1359 	/*
1360 	* Free the configuration structures
1361 	*/
1361
1362 	/*
1363 	* Free the configuration structures
1364 	*/
1364
1365 	/*
1366 	* Free the configuration structures
1367 	*/
1367
1368 	/*
1369 	* Free the configuration structures
1370 	*/
1370
1371 	/*
1372 	* Free the configuration structures
1373 	*/
1373
1374 	/*
1375 	* Free the configuration structures
1376 	*/
1376
1377 	/*
1378 	* Free the configuration structures
1379 	*/
1379
1380 	/*
1381 	* Free the configuration structures
1382 	*/
1382
1383 	/*
1384 	* Free the configuration structures
1385 	*/
1385
1386 	/*
1387 	* Free the configuration structures
1388 	*/
1388
1389 	/*
1390 	* Free the configuration structures
1391 	*/
1391
1392 	/*
1393 	* Free the configuration structures
1394 	*/
1394
1395 	/*
1396 	* Free the configuration structures
1397 	*/
1397
1398 	/*
1399 	* Free the configuration structures
1400 	*/
1400
1401 	/*
1402 	* Free the configuration structures
1403 	*/
1403
1404 	/*
1405 	* Free the configuration structures
1406 	*/
1406
1407 	/*
1408 	* Free the configuration structures
1409 	*/
1409
1410 	/*
1411 	* Free the configuration structures
1412 	*/
1412
1413 	/*
1414 	* Free the configuration structures
1415 	*/
1415
1416 	/*
1417 	* Free the configuration structures
1418 	*/
1418
1419 	/*
1420 	* Free the configuration structures
1421 	*/
1421
1422 	/*
1423 	* Free the configuration structures
1424 	*/
1424
1425 	/*
1426 	* Free the configuration structures
1427 	*/
1427
1428 	/*
1429 	* Free the configuration structures
1430 	*/
1430
1431 	/*
1432 	* Free the configuration structures
1433 	*/
1433
1434 	/*
1435 	* Free the configuration structures
1436 	*/
1436
1437 	/*
1438 	* Free the configuration structures
1439 	*/
1439
1440 	/*
1441 	* Free the configuration structures
1442 	*/
1442
1443 	/*
1444 	* Free the configuration structures
1445 	*/
1445
1446 	/*
1447 	* Free the configuration structures
1448 	*/
1448
1449 	/*
1450 	* Free the configuration structures
1451 	*/
1451
1452 	/*
1453 	* Free the configuration structures
1454 	*/
1454
1455 	/*
1456 	* Free the configuration structures
1457 	*/
1457
1458 	/*
1459 	* Free the configuration structures
1460 	*/
1460
1461 	/*
1462 	* Free the configuration structures
1463 	*/
1463
1464 	/*
1465 	* Free the configuration structures
1466 	*/
1466
1467 	/*
1468 	* Free the configuration structures
1469 	*/
1469
1470 	/*
1471 	* Free the configuration structures
1472 	*/
1472
1473 	/*
1474 	* Free the configuration structures
1475 	*/
1475
1476 	/*
1477 	* Free the configuration structures
1478 	*/
1478
1479 	/*
1480 	* Free the configuration structures
1481 	*/
1481
1482 	/*
1483 	* Free the configuration structures
1484 	*/
1484
1485 	/*
1486 	* Free the configuration structures
1487 	*/
1487
1488 	/*
1489 	* Free the configuration structures
1490 	*/
1490
1491 	/*
1492 	* Free the configuration structures
1493 	*/
1493
1494 	/*
1495 	* Free the configuration structures
1496 	*/
1496
1497 	/*
1498 	* Free the configuration structures
1499 	*/
1499
1500 	/*
1501 	* Free the configuration structures
1502 	*/
1502
1503 	/*
1504 	* Free the configuration structures
1505 	*/
1505
1506 	/*
1507 	* Free the configuration structures
1508 	*/
1508
1509 	/*
1510 	* Free the configuration structures
1511 	*/
1511
1512 	/*
1513 	* Free the configuration structures
1514 	*/
1514
1515 	/*
1516 	* Free the configuration structures
1517 	*/
1517
1518 	/*
1519 	* Free the configuration structures
1520 	*/
1520
1521 	/*
1522 	* Free the configuration structures
1523 	*/
1523
1524 	/*
1525 	* Free the configuration structures
1526 	*/
1526
1527 	/*
1528 	* Free the configuration structures
1529 	*/
1529
1530 	/*
1531 	* Free the configuration structures
1532 	*/
1532
1533 	/*
1534 	* Free the configuration structures
1535 	*/
1535
1536 	/*
1537 	* Free the configuration structures
1538 	*/
1538
1539 	/*
1540 	* Free the configuration structures
1541 	*/
1541
1542 	/*
1543 	* Free the configuration structures
1544 	*/
1544
1545 	/*
1546 	* Free the configuration structures
1547 	*/
1547
1548 	/*
1549 	* Free the configuration structures
1550 	*/
1550
1551 	/*
1552 	* Free the configuration structures
1553 	*/
1553
1554 	/*
1555 	* Free the configuration structures
1556 	*/
1556
1557 	/*
1558 	* Free the configuration structures
1559 	*/
1559
1560 	/*
1561 	* Free the configuration structures
1562 	*/
1562
1563 	/*
1564 	* Free the configuration structures
1565 	*/
1565
1566 	/*
1567 	* Free the configuration structures
1568 	*/
1568
1569 	/*
1570 	* Free the configuration structures
1571 	*/
1571
1572 	/*
1573 	* Free the configuration structures
1574 	*/
1574
1575 	/*
1576 	* Free the configuration structures
1577 	*/
1577
1578 	/*
1579 	* Free the configuration structures
1580 	*/
1580
1581 	/*
1582 	* Free the configuration structures
1583 	*/
1583
1584 	/*
1585 	* Free the configuration structures
1586 	*/
1586
1587 	/*
1588 	* Free the configuration structures
1589 	*/
1589
1590 	/*
1591 	* Free the configuration structures
1592 	*/
1592
1593 	/*
1594 	* Free the configuration structures
1595 	*/
1595
1596 	/*
1597 	* Free the configuration structures
1598 	*/
1598
1599 	/*
1599
1600 	* Free the configuration structures
1601 	*/
1601
1602 	/*
1603 	* Free the configuration structures
1604 	*/
1604
1605 	/*
1606 	* Free the configuration structures
1607 	*/
1607
1608 	/*
1609 	* Free the configuration structures
1610 	*/
1610
1611 	/*
1612 	* Free the configuration structures
1613 	*/
1613
1614 	/*
1615 	* Free the configuration structures
1616 	*/
1616
1617 	/*
1618 	* Free the configuration structures
1619 	*/
1619
1620 	/*
1621 	* Free the configuration structures
1622 	*/
1622
1623 	/*
1624 	* Free the configuration structures
1625 	*/
1625
1626 	/*
1627 	* Free the configuration structures
1628 	*/
1628
1629 	/*
1630 	* Free the configuration structures
1631 	*/
1631
1632 	/*
1633 	* Free the configuration structures
1634 	*/
163
```

339 1 /*
340 1 * Free the various simple string buffers
* ,

```

355 2
356 1
    }
357 1
    if (NULL != rcp->rc_human_uidname)
358 1
    {
359 2
        free(rcp->rc_human_uidname);
360 2
    }
361 1
}
362 1
if (NULL != rcp->rc_effective_uidname)
363 1
{
364 2
    /* don't free, its internal: free(rcp->rc_effective_uidname);
365 2
    */
418 1
rcp = NULL;
419 1
420 1
    return( err );
421 1
    /* RSTSL_Finish */
422 1
}
423 1

```

```

    init_plugins

427  /*
428  * init_plugins
429  *
430  * Function Description:
431  *
432  * This function locates, opens, validates and initializes all restore
433  * plug-in (shared) libraries. They must be located in
434  * /usr/epoch/EB/cure_plugin (
435  * directory are opened and validates for version# and presence of
436  * mandatory functions.
437  * library to determine which optional features are supported,
438  * the corresponding functions are present. Finally,
439  * function is called for each valid library.
440  *
441  * Parameters:
442  *
443  * Inputs:
444  *   rcp      (I) - Pointer to restore context
445  * Outputs:
446  *   none
447  *
448  * Returns:
449  *   E_SUCCESS or EP_RB_RECOVER_xxx
450  *
451  * Logic/pseudo code:
452  *
453  * open plugin dir
454  * while read next_entry succeeds
455  *   verify .so file (else continue)
456  *   open shared library file (else continue)
457  *   on errors below:
458  *     close shared library file
459  *     continue
460  *   fetch all mandatory function addresses
461  *   call identify function
462  *   validate version number
463  *   fetch all indicated optional function addrs
464  *   call Initialize function
465  *   add workitem types to composite exclusion list
466  *   add to valid plugin list
467  *   close plugin dir
468  *
469  */

```

```

static errno_ty init_plugins( restore_context *rcp )
{
    DIR      *dirp;
    struct dirent *direntp;
    errno_ty status = E_SUCCESS;
    struct pluginData *pidataPtr = NULL;
    struct pluginData *pilistPtr = NULL;
    int      val_result;
    struct pluginIDdata *iddataPtr;
    char    *tmp_types;
    int      shlib_dirlen;
    char    shlib_path [MAXPATHLEN];
    /* open plugin directory or bust */
    if ( NULL == (dirp = opendir( eb_cure_plugin_dir ) ) )
        rec_api_log_csm( SUB_CSM_PLUGIN_ERR, NULL );
    #if 1
    return E_SUCCESS; /* allow continuation w/o plugins */
    #else
    return EP_RB_RECOVER_NO_PLUGINS; /* later do this */
    #endif
}
    strcpy( shlib_path, eb_cure_plugin_dir );
    strcat( shlib_path, "/" );
    shlib_dirlen = strlen( shlib_path );
    /* loop thru entries in directory*/
    while (NULL != (direntp = readdir( dirp )))
    {
        if (NULL == piDataPtr)
        {
            /* allocate next plugin data structure */
            if (NULL == (piDataPtr
                         = calloc( 1, sizeof(
                           struct pluginData) )))
                status = EP_RB_RECOVER_NOMEM;
            break; /* fall thru to cleanup */
        }
        if (NULL == strstr( direntp->d_name, ".so" ) )
            continue; /* skip this guy */
        strcpy( &shlib_path[shlib_dirlen], direntp->d_name );
        if (NULL == (pidataPtr->libHdl
                     = dlopen( shlib_path, RTLD_NOW )))
        {
            rbe_user_error( 0, "Error opening restore plug-in library
                           %s: %s\n",
                           direntp->d_name, dlerror() );
            continue; /* skip this one */
        }
        /* Fetch addresses of all mandatory functions and */
        /* Do Identify processing: call it, save options, validate */
        if (0 != (val_result = validate_plugin(
                    piDataPtr )))
            continue;
        /* Functions missing from restore plug-in library %s:
         *   if (val_result == -1 || val_result == -4)
         *     rbe_user_error( 0,
         *                     direntp->d_name, dlerror() );
         *   else if (val_result < 0)
         *     rbe_user_error( 0,
         *                     "Validation failed for restore plug-in
         *                     library %s\n",
         *                     direntp->d_name );
         */
    }
}

```

```

542 3
543 4
544 4
545 4
546 4
547 3
548 3
549 3
550 3
551 3
552 2
553 2
554 2
555 2
556 2
557 2
558 3
559 3
560 3
561 3
562 3
563 3
564 3
565 3
566 2
567 2
568 2
569 2
570 2
571 2
572 2
573 2
574 2
575 2
576 2
577 2
578 2
579 2
580 2
581 2
582 2
583 3
584 3
585 3
586 4
587 4
588 4
589 3
590 3
591 4
592 4
593 4
594 4

    else
    {
        rbe_user_error( val_result,
                        "RSTPII_Identify failed for restore plug-in library
                         %s\n",
                        direntp->d_name );
    }

    /* let DC plug-in do its initialization */
    piDataPtr->libHdl = NULL;
    continue; /* on any error, skip this lib */
}

/* save plugin's appData */
piDataPtr->appData = rcp->appData;
rcp->appData = NULL;

/* add piDataPtr to valid plugin list */
if (NULL == pilistPtr)
    rcp->pilist= piDataPtr;
else
    pilistPtr->next = piDataPtr;
    /* first in list */
    pilistPtr = piDataPtr;
    /* new end of list */
piDataPtr = NULL;

/* add workitem types to composite exclusion list */
idDataPtr = (struct pluginidata *)pilistPtr->idData;
if (idDataPtr->num_types > 0)
{
    tmp_types = calloc( 1, 1 + idDataPtr->num_types
                       + rcp->rc_num_plugin_wi_types
                       );
    if (NULL == tmp_types) {
        status = EP_RB_NOMEM;
        break;
    }
    if (NULL != rcp->rc_plugin_wi_types)
    {
        /* move old list to new buffer and free old list */
        memcpy( tmp_types,
                rcp->rc_num_plugin_wi_types,
                rcp->rc_num_plugin_wi_types );
    }
}

```


723

/* validate_plugin */

```

1 #define _POSIX_SOURCE 1
3 #include <restore/restoretree.h>
4 #include <restore/RSIplugin.h>
5 #include <ebconfig/rbconfig.h>
8 //
```

```

10 extern "C" void dump_unix_time(RSRPC_time_ty time)
11 1 {
12 1   Datetime t(time);
13 1   if(time)
14 2   {
15 2     cout << "Time: " << t << endl ;
16 1   }
17 1   else
18 2   {
19 2     cout << "Time: Zero value\n";
20 1   }
21 }
```

```

23  extern "C" void dump_time_list(RSTRPC_time_list *list, ostream &out)
24  {
25    Datetime *t;
26    out << "Dump of time list\n";
27    for(RSTRPC_time_list *listelem=list; listelem;
28      listelem=listelem->next)
29    {
30      t=new Datetime(listelem->time);
31      out << " Time: " << *t << endl;
32      free (t);
33    }
34    out << "End of time list\n";
35  }
36
37 }

```

```

39  extern "C" void dump_tlo_list(RSTRPC_tlo_list *list, ostream &out)
40  {
41    out << "Dump of RSTRPC_tlo_list\n";
42    for(RSTRPC_tlo_list *tlo=list->next)
43    {
44      RSTRPC_top_level_obj *tobj=list->tlo;
45      RSTRPC_restoreable_obj_root *ro=&(tobj->root);
46      out << "TLO " << endl;
47      out << " level: " << ro->objLevel << " Backup App: " <<
48      ro->backupApp << endl;
49      if(ro->objName)
50      {
51        out << " name : " << ro->objName << endl;
52      }
53      if(ro->objTypeString)
54      {
55        out << " types: " << ro->objTypeString << endl;
56      }
57    }
58    out << "End of dump RSTRPC_tlo_list\n";
59  }
60

```

```
62 //  
63 extern "C" void dump_uro_list(RSTRPC_uro_list *list, ostream &out)  
64 {  
65     out << "Dump of RSTRPC_uro_list\n";  
66     for(;list;list=list->next)  
67     {  
68         2 out << "We have a node\n";  
69         1 }  
70         1 out << "End of dump RSTRPC_uro_list\n";  
71 }
```

```

73
74     eerrno_ty RSTPI_Initialize	restore_context *context
75     {
76
77     // See if an initial sleep is needed
78     /* char *sl=getenv("RSTPI_INITIALIZE_SLEEP");
79     if(!sl)
80     {
81     int st=atoi(sl);
82     rbe_log_stats(0,"RSTPI_Initialize, sleep of %d seconds",st);
83     Sleep(st);
84     }
85     else
86     {
87     rbe_log_stats(0,"RSTPI_Initialize, no delay on startup");
88     }
89     */
90     // Allocate restore context data
91     context->appData=(void *)malloc(sizeof(struct restoreContextData));
92
93     if(!context->appData)
94     {
95     rbe_log_stats(0,"plugin.cc - malloc failure");
96     return EP_RB_MALLOC_FAILURE;
97     }
98
99     struct restoreContextData *rcd=
100     (struct restoreContextData *) (context->appData);
101
102     rcd->currentWiseListNode=NULL;
103     rcd->currentWiseNode=NULL;
104     rcd->currentBackupNode=NULL;
105
106     return E_SUCCESS;
107
}

```

108

//

110

//

```

111 // ****
112 * Submit
113 *
114 * This function creates a submit object from the currently marked
115 * restorable objects. The ID of the created submit object is passed to
116 * EDMRST_Start to begin execution of the restore.
117 *
118 * Parameters:
119 *   context           (I) - Pointer to the restore context
120 *   hostName         (I) - host to restore to (only if inplace == False)
121 *   policy           (I) - The overwrite policy to use
122 *   inplace          (
123 *
124 *   directory        (I) - Flag if the restoral is to be in original locations
125 *   transport        (
126 *     submitObjIDptr(          (I) - directory to restore to
127 *     IO) - ID of the submit user object created to describe
128 *   progressCB       (
129 *     I) - pointer to callback function to report progress and
130 *   *
131 *   Note:
132 *     The Progress callback is currently not used. If the need
133 *     for this develops, the routines which use this argument
134 *     (mark, unmark, submit) must be enhanced.
135 *
136 * ****
137 errno_t RSTPI_Submit( restore_context
138 *                      *context,
139 *                      const char   *hostName,
140 *                      const OverwritePolicy *policy,
141 *                      const boolean_t  inplace,
142 *                      const char   *directory,
143 *                      const RestoreTransport *transport,
144 *                      int          *submitObjIDptr,
145 *                      RSTPI_SubmitProgressProc progressCB
146 *                      )
147 *
148 *
149 * // Get the restore context so we can find the app data
150 * // struct restoreContextData *rcd=(struct restoreContextData *)
151 * context->appData);
152 *
153 * if(rcd)
154 * {
155 *   return EP_RB_RECOVER_RC_APP_DATA_NULL;
156 * }
157 *
158 * /**
159 * // Get the current backup node from the top level object
160 * // if(rcd->currentBackupNode)
161 * {
162 *   return EP_RB_RECOVER_CURRENT_BACKUP_NODE_NULL;
163 * }
164 *
165 * BackupNode *cbu=rcd->currentBackupNode;
166 *
167 * if(cbu)
168 *   scode=cbu->fillSubmitObject(context,
169 *                                restore_dcpif(plugin.cc 11

```

```

190  ****
191  * Get Top Level Objects
192  *
193  * This function is called to retrieve the configurable backup
194  * items for network backups and work item sets for Symmetrix Connect
195  * which are restorable for the given client.
196  *
197  * It is a GOAL of this routine to return all objects ever backed
198  * up successfully. For network backups, it only looks in the config
199  * file for 'top level objects' of the given client.
200  *
201  * While the restore API will be called repeatedly to retrieve a
202  * number of items on each call, this plug-in call must retrieve the whole
203  * set of applicable backup objects.
204  * will manage the composite list of top level objects from all
205  * backup apps.
206  *
207  * Parameters:
208  *   context          (I) - Pointer to the restore context
209  *   sourceHost        (I) - the name of the host whose backups are being restored
210  *   toplevobjs       (O) - ptr to linked list of Top Level Objects
211  *   numberEntries    (O) - the real number of objects returned in the list
212  *
213  * Returns:
214  *   E_SUCCESS        on success
215  *   EP_RB_RECOVER_XXX on error
216  *
217  ****
218  errno_t RSTPL_GetToplevelobjects(restore_context *context,
219  const char *sourceHost,
220  struct RSTRPC_tlo_list
221  {
222  short *numberEntries)
223  {
224  // Now look through current backups and put on list if in
225  // Range
226  // struct restoreContextData *rcd=(struct restoreContextData *)
227  // restoreContextData *rcd=(struct restoreContextData *)
228  // context->appData);
229  if(!rcd)
230  {
231  rcd->currentWisetListNode=new WiSetListNode((char *)sourceHost);
232  }
233  if(!rcd->currentWisetListNode->populate(
234  // Populate this node only
235  // if(!rcd->currentWisetListNode->populate(
236  // restore_dcpipplugin.cc 13
237  // Page 29 of 92
238  // if(!rcd->currentWisetListNode->populate(
239  // restore_dcpipplugin.cc 13
240  // restore_dcpipplugin.cc 13
241  // if(!rcd->currentWisetListNode->populate(
242  {
243  return EP_RB_RECOVER_WISETLIST_NODE_POPULATE_ERR;
244  }
245  *numberEntries=0;
246  *topLevelObjs=NULL;
247  RSTRPC_tlo_list *previous=NULL;
248  for(RestoreNode *rn = rcd->currentWisetListNode->getFirstChild();
249  (*numberEntries)++;
250  rn=rn->getNextChild();
251  RSTRPC_tlo_list *new_list_elem=(RSTRPC_tlo_list *)malloc(sizeof(
252  if(!new_list_elem)
253  {
254  rn=rn->currentWisetListNode->getNextChild();
255  (*numberEntries)--;
256  }
257  RSTRPC_tlo_list *new_list_elem=(RSTRPC_tlo_list *)malloc(sizeof(
258  if(!new_list_elem)
259  {
260  rn=rn->currentWisetListNode->getNextChild();
261  (*numberEntries)--;
262  RSTRPC_tlo_list *new_list_elem=(RSTRPC_tlo_list *)malloc(sizeof(
263  if(!new_list_elem)
264  {
265  if(!new_list_elem)
266  {
267  rn=rn->currentWisetListNode->getNextChild();
268  (*numberEntries)--;
269  }
270  memset(new_top, 0, sizeof(struct RSTRPC_top_level_obj *)malloc(
271  (*
272  * We need to check to see if a network is restorable for this
273  * top level object. Since the client machines are the same for
274  * all backups of the same workitem (duh!) we only need to check
275  * one. Although this should always work, we will behave in a
276  * fail-safe manner by
277  * (a) Making sure the child is non-null
278  * (b) Marking network as "not possible" when we find that case
279  * (so we consider network restore possible, and an allowed
280  * option, if we don't find a client in the ddtab for the
281  * first backup of the work item)
282  * BackupNode *bn=(BackupNode *)rn->getFirstChild();
283  if(bn && !bn->getNetworkRestorePossible())
284  {
285  // C++ standard evaluates left to right, so this is null-safe
286  if(bn && !bn->getNetworkRestorePossible())
287  {
288  {
289  new_top->flags |= TLO_BITFLAG_NETWORK_RESTORE_NOT_POSSIBLE;
290  }
291  }
292  char ** temp_ptr;
293  temp_ptr=(char **)malloc(sizeof(char *));
294  if(!temp_ptr)
295  {
296  rbe_log_stats(0, "plugin.cc - malloc failure");
297  return EP_RB_RECOVER_MALLOC_FAILURE;
298  }
299  *temp_ptr=(char *)rn;
300  new_top->appData.data=(char *)temp_ptr;
301  new_top->appData.length=sizeof(char *);
302  }
303  }
304  }
305  }
306  }
307  }
308  }
309  }
310  }
311  }
312  }
313  }
314  }
315  }
316  }
317  }
318  }
319  }
320  }
321  }
322  }
323  }
324  }
325  }
326  }
327  }
328  }
329  }
330  }
331  }
332  }
333  }
334  }
335  }
336  }
337  }
338  }
339  }
340  }
341  }
342  }
343  }
344  }
345  }
346  }
347  }
348  }
349  }
350  }
351  }
352  }
353  }
354  }
355  }
356  }
357  }
358  }
359  }
360  }
361  }
362  }
363  }
364  }
365  }
366  }
367  }
368  }
369  }
370  }
371  }
372  }
373  }
374  }
375  }
376  }
377  }
378  }
379  }
380  }
381  }
382  }
383  }
384  }
385  }
386  }
387  }
388  }
389  }
390  }
391  }
392  }
393  }
394  }
395  }
396  }
397  }
398  }
399  }
400  }
401  }
402  }
403  }
404  }
405  }
406  }
407  }
408  }
409  }
410  }
411  }
412  }
413  }
414  }
415  }
416  }
417  }
418  }
419  }
420  }
421  }
422  }
423  }
424  }
425  }
426  }
427  }
428  }
429  }
430  }
431  }
432  }
433  }
434  }
435  }
436  }
437  }
438  }
439  }
440  }
441  }
442  }
443  }
444  }
445  }
446  }
447  }
448  }
449  }
450  }
451  }
452  }
453  }
454  }
455  }
456  }
457  }
458  }
459  }
460  }
461  }
462  }
463  }
464  }
465  }
466  }
467  }
468  }
469  }
470  }
471  }
472  }
473  }
474  }
475  }
476  }
477  }
478  }
479  }
480  }
481  }
482  }
483  }
484  }
485  }
486  }
487  }
488  }
489  }
490  }
491  }
492  }
493  }
494  }
495  }
496  }
497  }
498  }
499  }
500  }
501  }
502  }
503  }
504  }
505  }
506  }
507  }
508  }
509  }
510  }
511  }
512  }
513  }
514  }
515  }
516  }
517  }
518  }
519  }
520  }
521  }
522  }
523  }
524  }
525  }
526  }
527  }
528  }
529  }
530  }
531  }
532  }
533  }
534  }
535  }
536  }
537  }
538  }
539  }
540  }
541  }
542  }
543  }
544  }
545  }
546  }
547  }
548  }
549  }
550  }
551  }
552  }
553  }
554  }
555  }
556  }
557  }
558  }
559  }
560  }
561  }
562  }
563  }
564  }
565  }
566  }
567  }
568  }
569  }
570  }
571  }
572  }
573  }
574  }
575  }
576  }
577  }
578  }
579  }
580  }
581  }
582  }
583  }
584  }
585  }
586  }
587  }
588  }
589  }
590  }
591  }
592  }
593  }
594  }
595  }
596  }
597  }
598  }
599  }
600  }
601  }
602  }
603  }
604  }
605  }
606  }
607  }
608  }
609  }
610  }
611  }
612  }
613  }
614  }
615  }
616  }
617  }
618  }
619  }
620  }
621  }
622  }
623  }
624  }
625  }
626  }
627  }
628  }
629  }
630  }
631  }
632  }
633  }
634  }
635  }
636  }
637  }
638  }
639  }
640  }
641  }
642  }
643  }
644  }
645  }
646  }
647  }
648  }
649  }
650  }
651  }
652  }
653  }
654  }
655  }
656  }
657  }
658  }
659  }
660  }
661  }
662  }
663  }
664  }
665  }
666  }
667  }
668  }
669  }
670  }
671  }
672  }
673  }
674  }
675  }
676  }
677  }
678  }
679  }
680  }
681  }
682  }
683  }
684  }
685  }
686  }
687  }
688  }
689  }
690  }
691  }
692  }
693  }
694  }
695  }
696  }
697  }
698  }
699  }
700  }
701  }
702  }
703  }
704  }
705  }
706  }
707  }
708  }
709  }
710  }
711  }
712  }
713  }
714  }
715  }
716  }
717  }
718  }
719  }
720  }
721  }
722  }
723  }
724  }
725  }
726  }
727  }
728  }
729  }
730  }
731  }
732  }
733  }
734  }
735  }
736  }
737  }
738  }
739  }
740  }
741  }
742  }
743  }
744  }
745  }
746  }
747  }
748  }
749  }
750  }
751  }
752  }
753  }
754  }
755  }
756  }
757  }
758  }
759  }
760  }
761  }
762  }
763  }
764  }
765  }
766  }
767  }
768  }
769  }
770  }
771  }
772  }
773  }
774  }
775  }
776  }
777  }
778  }
779  }
770  }
771  }
772  }
773  }
774  }
775  }
776  }
777  }
778  }
779  }
780  }
781  }
782  }
783  }
784  }
785  }
786  }
787  }
788  }
789  }
780  }
781  }
782  }
783  }
784  }
785  }
786  }
787  }
788  }
789  }
790  }
791  }
792  }
793  }
794  }
795  }
796  }
797  }
798  }
799  }
800  }
801  }
802  }
803  }
804  }
805  }
806  }
807  }
808  }
809  }
801  }
802  }
803  }
804  }
805  }
806  }
807  }
808  }
809  }
810  }
811  }
812  }
813  }
814  }
815  }
816  }
817  }
818  }
819  }
811  }
812  }
813  }
814  }
815  }
816  }
817  }
818  }
819  }
820  }
821  }
822  }
823  }
824  }
825  }
826  }
827  }
828  }
829  }
830  }
831  }
832  }
833  }
834  }
835  }
836  }
837  }
838  }
839  }
831  }
832  }
833  }
834  }
835  }
836  }
837  }
838  }
839  }
840  }
841  }
842  }
843  }
844  }
845  }
846  }
847  }
848  }
849  }
841  }
842  }
843  }
844  }
845  }
846  }
847  }
848  }
849  }
850  }
851  }
852  }
853  }
854  }
855  }
856  }
857  }
858  }
859  }
851  }
852  }
853  }
854  }
855  }
856  }
857  }
858  }
859  }
860  }
861  }
862  }
863  }
864  }
865  }
866  }
867  }
868  }
869  }
861  }
862  }
863  }
864  }
865  }
866  }
867  }
868  }
869  }
870  }
871  }
872  }
873  }
874  }
875  }
876  }
877  }
878  }
879  }
871  }
872  }
873  }
874  }
875  }
876  }
877  }
878  }
879  }
880  }
881  }
882  }
883  }
884  }
885  }
886  }
887  }
888  }
889  }
881  }
882  }
883  }
884  }
885  }
886  }
887  }
888  }
889  }
890  }
891  }
892  }
893  }
894  }
895  }
896  }
897  }
898  }
899  }
891  }
892  }
893  }
894  }
895  }
896  }
897  }
898  }
899  }
900  }
901  }
902  }
903  }
904  }
905  }
906  }
907  }
908  }
909  }
901  }
902  }
903  }
904  }
905  }
906  }
907  }
908  }
909  }
910  }
911  }
912  }
913  }
914  }
915  }
916  }
917  }
918  }
919  }
911  }
912  }
913  }
914  }
915  }
916  }
917  }
918  }
919  }
920  }
921  }
922  }
923  }
924  }
925  }
926  }
927  }
928  }
929  }
930  }
931  }
932  }
933  }
934  }
935  }
936  }
937  }
938  }
939  }
931  }
932  }
933  }
934  }
935  }
936  }
937  }
938  }
939  }
940  }
941  }
942  }
943  }
944  }
945  }
946  }
947  }
948  }
949  }
941  }
942  }
943  }
944  }
945  }
946  }
947  }
948  }
949  }
950  }
951  }
952  }
953  }
954  }
955  }
956  }
957  }
958  }
959  }
951  }
952  }
953  }
954  }
955  }
956  }
957  }
958  }
959  }
960  }
961  }
962  }
963  }
964  }
965  }
966  }
967  }
968  }
969  }
961  }
962  }
963  }
964  }
965  }
966  }
967  }
968  }
969  }
970  }
971  }
972  }
973  }
974  }
975  }
976  }
977  }
978  }
979  }
971  }
972  }
973  }
974  }
975  }
976  }
977  }
978  }
979  }
980  }
981  }
982  }
983  }
984  }
985  }
986  }
987  }
988  }
989  }
981  }
982  }
983  }
984  }
985  }
986  }
987  }
988  }
989  }
990  }
991  }
992  }
993  }
994  }
995  }
996  }
997  }
998  }
999  }
991  }
992  }
993  }
994  }
995  }
996  }
997  }
998  }
999  }
1000  }
1001  }
1002  }
1003  }
1004  }
1005  }
1006  }
1007  }
1008  }
1009  }
1001  }
1002  }
1003  }
1004  }
1005  }
1006  }
1007  }
1008  }
1009  }
1010  }
1011  }
1012  }
1013  }
1014  }
1015  }
1016  }
1017  }
1018  }
1019  }
1011  }
1012  }
1013  }
1014  }
1015  }
1016  }
1017  }
1018  }
1019  }
1020  }
1021  }
1022  }
1023  }
1024  }
1025  }
1026  }
1027  }
1028  }
1029  }
1030  }
1031  }
1032  }
1033  }
1034  }
1035  }
1036  }
1037  }
1038  }
1039  }
1031  }
1032  }
1033  }
1034  }
1035  }
1036  }
1037  }
1038  }
1039  }
1040  }
1041  }
1042  }
1043  }
1044  }
1045  }
1046  }
1047  }
1048  }
1049  }
1041  }
1042  }
1043  }
1044  }
1045  }
1046  }
1047  }
1048  }
1049  }
1050  }
1051  }
1052  }
1053  }
1054  }
1055  }
1056  }
1057  }
1058  }
1059  }
1051  }
1052  }
1053  }
1054  }
1055  }
1056  }
1057  }
1058  }
1059  }
1060  }
1061  }
1062  }
1063  }
1064  }
1065  }
1066  }
1067  }
1068  }
1069  }
1061  }
1062  }
1063  }
1064  }
1065  }
1066  }
1067  }
1068  }
1069  }
1070  }
1071  }
1072  }
1073  }
1074  }
1075  }
1076  }
1077  }
1078  }
1079  }
1071  }
1072  }
1073  }
1074  }
1075  }
1076  }
1077  }
1078  }
1079  }
1080  }
1081  }
1082  }
1083  }
1084  }
1085  }
1086  }
1087  }
1088  }
1089  }
1081  }
1082  }
1083  }
1084  }
1085  }
1086  }
1087  }
1088  }
1089  }
1090  }
1091  }
1092  }
1093  }
1094  }
1095  }
1096  }
1097  }
1098  }
1099  }
1091  }
1092  }
1093  }
1094  }
1095  }
1096  }
1097  }
1098  }
1099  }
1100  }
1101  }
1102  }
1103  }
1104  }
1105  }
1106  }
1107  }
1108  }
1109  }
1101  }
1102  }
1103  }
1104  }
1105  }
1106  }
1107  }
1108  }
1109  }
1110  }
1111  }
1112  }
1113  }
1114  }
1115  }
1116  }
1117  }
1118  }
1119  }
1111  }
1112  }
1113  }
1114  }
1115  }
1116  }
1117  }
1118  }
1119  }
1120  }
1121  }
1122  }
1123  }
1124  }
1125  }
1126  }
1127  }
1128  }
1129  }
1130  }
1131  }
1132  }
1133  }
1134  }
1135  }
1136  }
1137  }
1138  }
1139  }
1131  }
1132  }
1133  }
1134  }
1135  }
1136  }
1137  }
1138  }
1139  }
1140  }
1141  }
1142  }
1143  }
1144  }
1145  }
1146  }
1147  }
1148  }
1149  }
1141  }
1142  }
1143  }
1144  }
1145  }
1146  }
1147  }
1148  }
1149  }
1150  }
1151  }
1152  }
1153  }
1154  }
1155  }
1156  }
1157  }
1158  }
1159  }
1151  }
1152  }
1153  }
1154  }
1155  }
1156  }
1157  }
1158  }
1159  }
1160  }
1161  }
1162  }
1163  }
1164  }
1165  }
1166  }
1167  }
1168  }
1169  }
1161  }
1162  }
1163  }
1164  }
1165  }
1166  }
1167  }
1168  }
1169  }
1170  }
1171  }
1172  }
1173  }
1174  }
1175  }
1176  }
1177  }
1178  }
1179  }
1171  }
1172  }
1173  }
1174  }
1175  }
1176  }
1177  }
1178  }
1179  }
1180  }
1181  }
1182  }
1183  }
1184  }
1185  }
1186  }
1187  }
1188  }
1189  }
1181  }
1182  }
1183  }
1184  }
1185  }
1186  }
1187  }
1188  }
1189  }
1190  }
1191  }
1192  }
1193  }
1194  }
1195  }
1196  }
1197  }
1198  }
1199  }
1191  }
1192  }
1193  }
1194  }
1195  }
1196  }
1197  }
1198  }
1199  }
1200  }
1201  }
1202  }
1203  }
1204  }
1205  }
1206  }
1207  }
1208  }
1209  }
1201  }
1202  }
1203  }
1204  }
1205  }
1206  }
1207  }
1208  }
1209  }
1210  }
1211  }
1212  }
1213  }
1214  }
1215  }
1216  }
1217  }
1218  }
1219  }
1220  }
1221  }
1222  }
1223  }
1224  }
1225  }
1226  }
1227  }
1228  }
1229  }
1230  }
1231  }
1232  }
1233  }
1234  }
1235  }
1236  }
1237  }
1238  }
1239  }
1240  }
1241  }
1242  }
1243  }
1244  }
1245  }
1246  }
1247  }
1248  }
1249  }
1250  }
1251  }
1252  }
1253  }
1254  }
1255  }
1256  }
1257  }
1258  }
1259  }
1251  }
1252  }
1253  }
1254  }
1255  }
1256  }
1257  }
1258  }
1259  }
1260  }
1261  }
1262  }
1263  }
1264  }
1265  }
1266  }
1267  }
1268  }
1269  }
1270  }
1271  }
1272  }
1273  }
1274  }
1275  }
1276  }
1277  }
1278  }
1279  }
1280  }
1281  }
1282  }
1283  }
1284  }
1285  }
1286  }
1287  }
1288  }
1289  }
1290  }
1291  }
1292  }
1293  }
1294  }
1295  }
1296  }
1297  }
1298  }
1299  }
1300  }
1301  }
1302  }
1303  }
1304  }
1305  }
1306  }
1307  }
1308  }
1309  }
1310  }
1311  }
1312  }
1313  }
1314  }
1315  }
1316  }
1317  }
1318  }
1319  }
1320  }
1321  }
1322  }
1323  }
1324  }
1325  }
1326  }
1327  }
1328  }
1329  }
1330  }
1331  }
1332  }
1333  }
1334  }
1335  }
1336  }
1337  }
1338  }
1339  }
1340  }
1341  }
1342  }
1343  }
1344  }
1345  }
1346  }
1347  }
1348  }
1349  }
1350  }
1351  }
1352  }
1353  }
1354  }
1355  }
1356  }
1357  }
1358  }
1359  }
1360  }
1361  }
1362  }
1363  }
1364  }
1365  }
1366  }
1367  }
1368  }
1369  }
1370  }
1371  }
1372  }
1373  }
1374  }
1375  }
1376  }
1377  }
1378  }
1379  }
1380  }
1381  }
1382  }
1383  }
1384  }
1385  }
1386  }
1387  }
1388  }
1389  }
1390  }
1391  }
1392  }
1393  }
1394  }
1395  }
1396  }
1397  }
1398  }
1399  }
1400  }
1401  }
1402  }
1403  }
1404  }
1405  }
1406  }
1407  }
1408  }
1409  }
1410  }
1411  }
1412  }
1413  }
1414  }
1415  }
1416  }
1417  }
1418  }
1419  }
1420  }
1421  }
1422  }
1423  }
1424  }
1425  }
1426  }
1427  }
1428  }
1429  }
1430  }
1431  }
1432  }
1433  }
1434  }
1435  }
1436  }
1437  }
1438  }
1439  }
1440  }
1441  }
1442  }
1443  }
1444  }
1445  }
1446  }
1447  }
1448  }
1449  }
1450  }
1451  }
1452  }
1453  }
1454  }
1455  }
1456  }
1457  }
1458  }
1459  }
1460  }
1461  }
1462  }
1463  }
1464  }
1465  }
1466  }
1467  }
1468  }
1469  }
1470  }
1471  }
1472  }
1473  }
1474  }
1475  }
1476  }
1477  }
1478  }
1479  }
1480  }
1481  }
1482  }
1483  }
1484  }
1485  }
1486  }
1487  }
1488  }
1489  }
1490  }
1491  }
1492  }
1493  }
1494  }
1495  }
1496  }
1497  }
1498  }
1499  }
1500  }
1501  }
1502  }
1503  }
1504  }
1505  }
1506  }
1507  }
1508  }
1509  }
1510  }
1511  }
1512  }
1513  }
1514  }
1515  }
1516  }
1517  }
1518  }
1519  }
1520  }
1521  }
1522  }
1523  }
1524  }
1525  }
1526  }
1527  }
1528  }
1529  }
1530  }
1531  }
1532  }
1533  }
1534  }
1535  }
1536  }
1537  }
1538  }
1539  }
1540  }
1541  }
1542  }
1543  }
1544  }
1545  }
1546  }
1547  }
1548  }
1549  }
1550  }
1551  }
1552  }
1553  }
1554  }
1555  }
1556  }
1557  }
1558  }
1559  }
1560  }
1561  }
1562  }
1563  }
1564  }
1565  }
1566  }
1567  }
1568  }
1569  }
1570  }
1571  }
1572  }
1573  }
1574  }
1575  }
1576  }
1577  }
1578  }
1579  }
1570  }
1571  }
1572  }
1573  }
1574  }
1575  }
1576  }
1577  }
1578  }
1579  }
1580  }
1581  }
1582  }
1583  }
1584  }
1585  }
1586  }
1587  }
1588  }
1589  }
1590  }
1591  }
1592  }
1593  }
1594  }
1595  }
1596  }
1597  }
1598  }
1599  }
1600  }
1601  }
1602  }
1603  }
1604  }
1605  }
1606  }
1607  }
1608  }
1609  }
1610  }
1611  }
1612  }
1613  }
1614  }
1615  }
1616  }
1617  }
1618  }
1619  }
1620  }
1621  }
1622  }
1623  }
1624  }
1625  }
1626  }
1627  }
1628  }
1629  }
1630  }
1631  }
1632  }
1633  }
1634  }
1635  }
1636  }
1637  }
1638  }
1639  }
1640  }
1641  }
1642  }
1643  }
1644  }
1645  }
1646  }
1647  }
1648  }
1649  }
1650  }
1651  }
1652  }
1653  }
1654  }
1655  }
1656  }
1657  }
1658  }
1659  }
1660  }
1661  }
1662  }
1663  }
1664  }
1665  }
1666  }
1667  }
1668  }
1669  }
1670  }
1671  }
1672  }
1673  }
1674  }
1675  }
1676  }
1677  }
1678  }
1679  }
1680  }
1681  }
1682  }
1683  }
1684  }
1685  }
1686  }
1687  }
1688  }
1689  }
1690  }
1691  }
1692  }
1693  }
1694  }
1695 
```

```

305 2     new_top->root.objLevel=RSTRPC_tlo_type;
306 2     new_top->root.objName=esl_strdup((char *) (rn->getNameChar ()));
307 2     if (NULL==new_top->root.objName)
308 3     {
309 3         rbe_log_stats(
310 3             0, "plugin.cc - allocation failure from esl_strdup failure");
311 2     }
312
313 2     new_top->wiBIC=NULL;
314 2     // new_top->templateName=esl_strdup (""); // Blank as placeholder
315 2
316 2     new_top->root.objTypeString=esl_strdup (""); // Blank as placeholder
317 2
318 2     new_top->hostname=esl_strdup (rcd->currentWisetListNode->getName());
319 2
320 2     new_top->fileSpec=esl_strdup (""); // Blank as placeholder
321 2
322 2     if (NULL == new_top->root.objTypeString ||
323 2         NULL == new_top->fileSpec)
324 3     {
325 3         rbe_log_stats(
326 3             0, "plugin.cc - allocation failure from esl_strdup");
327 2     }
328
329 2     new_top->wiType='0';
330 2     new_top->ssThread=FALSE;
331
332 2     new_list_elem->tlo=new_top;
333 2
334 2     if (previous)
335 2     {
336 3         previous->next=new_list_elem;
337 3     }
338 2
339 2     else
340 3     {
341 3         *topLevelObjs=new_list_elem;
342 2     }
343 2
344 2     previous=new_list_elem;
345 1
346
347 1     return E_SUCCESS;
348

```



```

528  ****
529  *      Get Next Level Objects:
530  *
531  *      This function is intended to allow retrieval of the children
532  *      of a given parent object. The caller specifies the parent object and
533  *      whether or not to include bad files. Even though the objects are
534  *      returned in a linked list, there could conceivably be thousands of
535  *      child objects, so the caller must specify the maximum number
536  *      of children to return. The caller is returned a token ( cookie ) to allow
537  *      continuing on the next call to this function.
538  *
539  *      Parameters:
540  *          context          (I) - Pointer to the restore context
541  *          parentObject    (I) - the parent object
542  *          objectLevel     (I) - specifies whether parentObject is a top level or
543  *          objects          (I) - a pointer to receive the start of the objects list
544  *          cookie          (IO) - a place holder for the list position
545  *          maxEntries      (I) - the maximum number of objects to return
546  *
547  *          numberEntries   (I) - the real number of objects returned in the list
548  *          allowBadFiles   (I) - flag whether or not to include bad files
549  *
550  *      Returns:
551  *          E_SUCCESS        on success
552  *          EP_RB_RECOVER_XXX on error
553  *          ****
554  *
555  *      errno_t RSTPL_GetNextLevelObjects	restore_context *context,
556  *      restorableObjectPtr parentObj,
557  *      enum RSTRPC_ObjectLevel          objectLevel,
558  *      struct RSTRPC_uro_list **objects,
559  *      long *cookie,
560  *      const long maxEntries,
561  *      const boolean_ty allowBadFiles)
562  *
563  {
564  *
565  *      struct restoreContextData *rcd=
566  *      (struct restoreContextData *) (context->appData);
567  *
568  *      if(rcd)
569  *      {
570  *          return EP_RB_RECOVER_RC_APP_DATA_NULL;
571  *      }
572  *
573  *      /**
574  *      // Geta the restore node of the child
575  *      /**
576  *      // RestoreNode *rnodep;
577  *
578  *      switch(objectLevel)
579  *      {
580  *          char ** c1;
581  *          switch(objectLevel)
582  *          {
583  *              case RSTRPC_tlo_type:
584  *                  c1=(char *) ((RSTRPC_top_level_obj *)parentObj)->appData.data);
585  *                  rnodep=(RestoreNode *) (c1);
586  *                  break;
587  *              case RSTRPC_container_type:
588  *          }
589  *      }
590  *      rnnodep=(RestoreNode *) (*c1);
591  *      break;
592  *      return EP_RB_RECOVER_BAD_OBJECT_LEVEL;
593  *  }
594  *  break;
595  * }
596  *
597  * if(!rnnodep)
598  * {
599  *     2
600  *     1
601  *     // If we are getting the children of the current workitem set node,
602  *     1
603  *     // is the one which pertains to the point in time we are interested
604  *     1
605  *     // actually want the children of the current backup node. The current backup node
606  *     1
607  *     // we
608  *     1
609  *     2
610  *     2
611  *     1
612  *     if(rnnodep==rcd->currentWisetNode)
613  *     1
614  *     2
615  *     2
616  *     1
617  *     if(!rnnodep)
618  *     1
619  *     1
620  *     int entries_count=0;
621  *     1
622  *     // This code checks to see if we have a cookie. If the cookie is non-null,
623  *     1
624  *     // it is actually a pointer to the position int he chain where we
625  *     1
626  *     // off.
627  *     1
628  *     // into a cookie ( long int).
629  *     1
630  *     1
631  *     2
632  *     2
633  *     1
634  *     1
635  *     2
636  *     2
637  *     2
638  *     1
639  *     RSTRPC_uro_list *previous=NULL;
640  *     1
641  *     1
642  *     1
643  *     RSTRPC_uro_list *listentry;
644  *     1
645  *     1
646  *     1
647  *     // Traverse the children of the current node and create a restorable
648  *     1
649  *     // object list.
650  *
651  * Thu Jan 03 12:52:58 2008      restore_dcpl/plugin.cc 21
652  * Page 37 of 92

```

```

Page 39 of 92 RSTPL_GeNextLevelObjects Thu Jan 03 12:52:58 2008
Page 40 of 92 RSTPL_GetNextLevelObjects Thu Jan 03 12:52:58 2008

647 1 // for(;child;child=rnodep->getNextChild())
648 1 {
649 2   // Skip this entry if we have chosen not to allow bad files
650 2   if( (FALSE == allowBadFiles) && (0 != rnodep->getStateBit(STATE_BADFILE) ) )
651 2   {
652 2     continue;
653 2   }
654 2   entries_count++;
655 3 }

656 3 listentry=(RSTRPC_uro_list *)malloc(sizeof(
657 2   struct RSTRPC_uro_list));
658 2   rbe_log_stats(0,"plugin.cc - malloc failure");
659 2   return EP_RB_RECOVER_MALLOC_FAILURE;
660 2 }

661 2 listentry=(RSTRPC_uro_list *)malloc(sizeof(
662 2   struct RSTRPC_uro_list));
663 3 if(!listentry)
664 3 {
665 3   rbe_log_stats(0,"plugin.cc - allocation failure from esl_strdup");
666 2 }

667 2 urobj=(RSTRPC_user_restorable_object *)malloc(sizeof(
668 2   struct RSTRPC_user_restorable_object));
669 2 if(!urobj)
670 3 {
671 3   rbe_log_stats(0,"plugin.cc - malloc failure");
672 3   return EP_RB_RECOVER_MALLOC_FAILURE;
673 2 }

674 2 memset(urobj,0,sizeof(RSTRPC_user_restorable_object));
675 2

676 2 listentry->uro=urobj;
677 2

678 2 char **temp_ptr;
679 2 temp_ptr=(char **)malloc(sizeof(char *));
680 2 if(!temp_ptr)
681 3 {
682 3   rbe_log_stats(0,"plugin.cc - malloc failure");
683 3   return EP_RB_RECOVER_MALLOC_FAILURE;
684 2 }

685 2 *temp_ptr=(char *)child;
686 2

687 2 urobj->appData.data=(char *)temp_ptr;
688 2 urobj->appData.length(sizeof(char *));
689 2

690 2 urobj->root.objLevel=child->getStateBit(
691 2   STATE_LEAFNODE)?RSTRPC_Leaf_type:RSTRPC_container_type;
692 2

693 2 char *child_name=(char *)child->getNameChar();
694 2 urobj->root.objName=esl_strdup(child_name);
695 2

696 3 if(NULL==urobj->root.objName)
697 3 {
698 3   rbe_log_stats(
699 2     0,"plugin.cc - allocation failure from esl_strdup");
699 2 }

700 2 // Find base name - token after final '/' if present,
701 2 // this final character.
702 2 // If the final character is a '/', look for the token preceeding
703 2

704 2 char *rp=NULL;
705 2 char *ch=NULL;
706 2 RnProperty *gname=child->getProperty(PROPERTY_GNAME);
707 2

Page 39 of 92 restore_dcpl/plugin.cc 23 Thu Jan 03 12:52:58 2008
Page 40 of 92 restore_dcpl/plugin.cc 24 Thu Jan 03 12:52:58 2008

```

```

773 2
774 3
775 3
776 3
777 2
780 2
781 2
782 3
783 3
784 2
785 2
786 3
787 3
788 3
789 3
790 4
791 4
792 4
793 5
794 5
795 4
796 3
797 2
799 2
800 2
802 2
803 3
804 3
805 2
806 2
807 2
808 2
809 3
810 3
811 2
812 2
813 2
814 2
815 2
816 3
817 3
818 2
819 2
820 3
821 3
822 2
824 2
826 2
827 3
828 3
829 2
830 2
831 3
832 3
833 2
834 2
    if(NULL==urobj->objOwnerName || NULL==urobj->objGroupName)
    {
        rbe_log_stats(0,"plugin.cc - allocation failure from esl_strdup");
        return EP_RB_RECOVER_MALLOC_FAILURE;
    }
    RnPropertyDate *pr=(RnPropertyDate *) (child->GetProperty(
        PROPERTY_BACKUP_DATE));
    if(pr)
    {
        urobj->objModTime=pr->getValue()->unixTime();
    }
    else
    {
        urobj->objModTime=0;
        BackupNode *bnode=(BackupNode *) (child->getBackupNodePtr());
        if(bnode)
        {
            pr=(RnPropertyDate *) (bnode->GetProperty(
                PROPERTY_BACKUP_DATE));
            if(pr)
            {
                urobj->objModTime=pr->getValue()->unixTime();
            }
        }
    }
    urobj->objSize.high=child->getSize().high;
    urobj->objSize.low=child->getSize().low;
    if(child->getStateBit(STATE_BADFILE))
    {
        urobj->objBackupStatus=RSTRPC_Backup_Bad;
    }
    else if( child->getStateBit(STATE_INVALID_SSID |
        STATE_NO_SSID |
        STATE_EXPIRED_SSID ) )
    {
        urobj->objBackupStatus=RSTRPC_Backup_Expired;
    }
    else if( child->getStateBit(STATE_CH_INVALID_SSID |
        STATE_CH_NO_SSID |
        STATE_CH_EXPIRED_SSID ) )
    {
        urobj->objBackupStatus=RSTRPC_Backup_Child_Without_Data;
    }
    else
    {
        urobj->objBackupStatus=RSTRPC_Backup_Good;
    }
    listentry->next=NULL;
    if(previous)
    {
        previous->next=listentry;
    }
    else
    {
        *objects=listentry;
    }
    previous=listentry;
}

```

```

836 2
837 3
838 3
839 3
840 3
841 2
842 1
844 1
845 1
846 1
847 1
    *cookie=DONE_COOKIE;
    *numberEntries=entries_count;
    return E_SUCCESS;
}

```

```

849 // ****
850
851 /* Mark Object
852 *
853 * The MarkObject operation takes a restorableObject and marks it, and
854 * possibly its descendant files for restoral based on the input
855 * criteria.
856 * Since the RSTSL_MarkObject call is an asynchronously executed
857 * in the Restore Engine that performs the marking, this function must
858 * periodically check for user-signalled cancellation, and update progress
859 * data using the progress callback function argument.
860 *
861 * NOTE: This functions is responsible for keeping the volumes needed
862 * (ebvlist) element of the restore context up to date.
863 *
864 * Parameters:
865 *
866 * context (I) - Pointer to the restore context
867 * thisObject (I) - The restoral object; can be a leaf object (e.g. a
868 * file), or a container object (e.g., a directory).
869 * allowBadfiles (I) - allows marking of files of state BADDATA.
870 * descend (
871 * * mark operation descend to operate on the content
872 * * of container objects.
873 * * BadFilesCount (O) - returns the file count with BADDATA.
874 * * PermDenyFilesCount (O) -- returns the file count with permission denied.
875 * * fileMarked (
876 * * lenMarkedFiles(
877 * * dirmarked (O) - return the total files marked after this mark
878 * * othermarked (
879 * * progressCB (
880 * * - pointer to callback function to report progress and
881 * * test for cancellation
882 * *
883 * * Note: The Progress callback is currently not used. If the need
884 * * for this develops, the routines which use this argument
885 * * (mark, unmark, submit) must be enhanced.
886 *
887 *
888 ****
889
890 errno_ty RSTPI_MarkObject(restore_context *context,
891 struct RSTRPC_user_restorable_object *thisObject,
892
893 boolean_ty allowBadFiles,
894
895 boolean_ty descend,
896 unsigned long *badFilesCount,
897 unsigned long *permDenyFilesCount,
898 unsigned long *filesMarked,
899 u_hyper *lenMarkedFiles,

```

```

900 1         unsigned long *dirMarked,
901 1         unsigned long *otherMarked,
902 1         RSTPI_MarkProgressProc progressCB)
903 1 {
904 1     char ** c1=(char **) (thisObject->appData.data);
905 1     RestoreNode *rnodep=(RestoreNode *) (*c1);
906 1     if(NULL==rnodep)
907 1     {
908 2         rbe_log_stats(
909 2             0, "RSTPI_MarkObject - Mark object has no app data");
910 2     }
911 1 }

913 1     /*
914 1      * MTFB (multi trail file backup) nodes are to be treated as atomic.
915 1      * If the item we are trying to mark is a child of a MTFB node, then
916 1      * we perform the mark operation on the entire MTFB node.
917 1
918 1     RestoreNode *parent=rnodep->getParent();
919 1     if(NULL !=parent && parent->nodeType() == RNC_MTFB)
920 2     {
921 2         rnodep=parent;
922 1     }

924 1     *badFilesCount=0;
925 1     *permDenyFilesCount=0;
926 1     *filesMarked=0;
927 1     *lMarkedFiles=ul_to_uh(0);
928 1     *dirMarked=0;
929 1     *otherMarked=0;

931 1     rnodep->markNode(descend,allowBadFiles);

933 1     /*
934 1      * We start at the backup node pointer since we want the total
935 1      * for the entire backup, not just this node down (except the
936 1      * badFilesCount, in which case the affected nodes will only
937 1      * be from this level down)
938 1
939 1     *fileMarked=rnodep->getBackupNodePtr()->countMarkedNodes(
940 1                                         RNC_ANY_DATAFILE);
941 1     *lenMarkedFiles=rnodep->getBackupNodePtr()->totalMarkedSize();

942 1     /*
943 1      * The bad file count is a count of the bad files encountered
944 1      * (not necessarily marked) from the current node down.
945 1
946 1     *badFilesCount=rnodep->countMarkedBadfileNodes(RNC_ANY_DATAFILE);
947 1
948 1
}

```

```

951 //*****
952 * UnmarkObject
953 *
954 * The UnmarkObject operation takes a restorableObject and unmarks
955 * possibly its descendant files for restoral based on the input
956 * criteria. Since the RSTSL_UmmarkObject call is an asynchronously executed
957 * in the Restore Engine that performs the unmarking, this plug-in function
958 * must periodically check for user-signalled cancellation, and update
959 * progress data using the progress callback function argument.
960 *
961 * NOTE: This functions is responsible for keeping the volumes needed
962 * (ebvlist) element of the restore context up to date.
963 *
964 * UnmarkObject Parameters:
965 *
966 * context (I) - The restoral object;
967 * thisObject (I) - can be a leaf object (e.g. a
968 * file), or a container object (e.g., a directory).
969 *
970 * BadFilesOnly (I) - allows unmarking ONLY of files of state BADDATA.
971 * descend (I) - Should unmark operation descend to operate on the
972 * content of container objects.
973 * BadFilesCount (O) - returns the file count with BADDATA.
974 * fileMarked (O) - return the total files marked after this unmark
975 * lenMarkedFiles (O) - return the length of files marked after this unmark
976 * otherMarked (O) - return the total directories marked after this unmark
977 * progressCB (I) - return the total "other" files marked after this unmark
978 * dirMarked (I) - pointer to callback function to report progress and
979 * otherMarked (I) - pointer to callback function to report progress and
980 * Note: The Progress Callback is currently not used. If the need
981 * For this develops, the routines which use this argument
982 * (mark, unmark, submit) must be enhanced.
983 *
984 *****/
985
986 errno_ty RSTPI_UmmarkObject(restore_context *context,
987 struct RSTRPC_user_restorable_object
988 *thisObject,
989 const boolean_ty BadFileOnly,
990 const boolean_ty descend,
991 unsigned long *BadFilesCount,
992 unsigned long *filesMarked,
993 u_hyper *lenMarkedFiles,
994 unsigned long *dirMarked,
995 unsigned long *otherMarked,
996 RSTPI_MarkProgressProc ProgressCB)
997 1 {
998 1   *BadFilesCount=0;
999 1   *filesMarked=0;
1000 1   *lenMarkedFiles=ul_to_uh(0);

```

```

1046 // ****
1047 * IS Object Markable
1048 *
1049 * This function determines if a restorable object has been
1050 * marked for restoration. It is intended to allow the user to determine the
1051 * current restore markings for the restorable objects at the same
1052 * hierarchy level,
1053 * i.e. objects that have the same parent restorableObject.
1054 *
1055 * Parameters:
1056 * context
1057 * thisObject (I) - Pointer to the restore context
1058 * thisObject (I) - The restorable object to be checked: can be a leaf object
1059 * (e.g. a file), or a container object (e.g., a directory).
1060 * markable (O) - boolean to receive the marked(I) / unmarked(O) result
1061 ****
1062
1063 errno_ty RSTPI_IsObjectMarkable	restore_context *context,
1064 struct RSTRPC_user_restoreable_object
1065 *thisObject,
1066 boolean_ty *markable)
1067
1068 1 {
1069 1 RestoreNode *rnodep = *(RestoreNode **)(thisObject->appdata.data);
1070 1 if (NULL==rnodep)
1071 2 {
1072 2 1 return EP_RB_RECOVER_RN_APP_DATA_NULL;
1073 1 }
1074 1 *markable=rnodep->isNodeMarkable();
1075 1
1076 1 return E_SUCCESS;
1077 1
1078 1
1079 1

```

```

1082 // ****
1083 * Is Object Marked
1084 *
1085 * This function determines if a restorable object has been
1086 * marked for restoration. It is intended to allow the user to determine the
1087 * current restore markings for the restorable objects at the same
1088 * object tree
1089 * hierarchy level,
1090 * i.e. objects that have the same parent restorableObject.
1091 * Parameters:
1092 * context
1093 * thisObject (I) - pointer to the restore context
1094 * thisObject (I) - The restoral object to be checked: can be a leaf object
1095 * (e.g. a file), or a container object (e.g., a directory).
1096 * marked (O) - boolean to receive the marked(I) / unmarked(0) result
1097 ****
1098 errno_ty RSTPL_IsObjectMarked	restore_context *context,
1099 struct RSTRPC_user_restorable_object
1100 *thisObject,
1101 boolean_ty *marked)
1102 {
1103 RestoreNode *rnodep = *(RestoreNode **) (thisObject->appData.data);
1104 if (NULL==rnodep)
1105 {
1106 1
1107 2
1108 2
1109 1
1110
1111 1
1112 *marked=rnodep->isNodeMarked();
1113 1
1114 return E_SUCCESS;
1115 }

```

1117 //

```

1118 // ****
1119 *
1120 * Get All Backup Times
1121 *
1122 * Function Description:
1123 *   Retrieve the dates of the backups within the time range
1124 *   specified by the caller.
1125 *
1126 * Parameters:
1127 *   context - (I) Pointer to the restore context
1128 *   startTime - (I) include no earlier than this date
1129 *   endTime - (I) include no later than this date
1130 *   flags - (I) flags - complete/partial
1131 *   timesList - (O) ptr to linked list of times
1132 *   numEntries - (O) count of times returned
1133 *
1134 * Return Codes:
1135 *   E_SUCCESS - operation completed successfully
1136 *   ****
1137

```

```

1138 errno_ty RSTPI_GetAllBackupTimes	restore_context *context,
1139 const time_t startTime,
1140 const time_t endTime,
1141 RSTRPC_backupFlags_ty flags,
1142 RSTRPC_timeList **timesList,
1143 short *numEntries)
1144 {

```

```

1145     *
1146     *timesList=NULL;
1147

```

```

1148     //
1149     // Now look through current backups and put on list if in
1150     // Range
1151     //
1152     // Range
1153     struct restoreContextData *rcd=(struct restoreContextData *)
1154     context->appData);

```

```

1155     if(!rcd)
1156     {
1157         return EP_RB_RECOVER_RC_APP_DATA_NULL;
1158     }

```

```

1159     WiSetNode *wilist=rcd->currentWiSetNode;
1160

```

```

1161     if(!wilist)
1162     {
1163         return EP_RB_RECOVER_RN_APP_DATA_NULL;
1164     }
1165

```

```

1166     int first=1;
1167

```

```

1168     RSTRPC_timeList *newlist=NULL;
1169
1170     *numEntries=0; // Count of number of entries found

```

```

1171     for(RestoreNode *bn=wilist->getFirstChild();bn=
1172         wilist->getNextChild())
1173     {
1174         RnPropertyDate *pr=(RnPropertyDate *) (bn->getProperty(
1175             PROPERTY_BACKUP_DATE));
1176         if(!pr)
1177         {
1178             continue;
1179         }

```

```

1180 2
1181 2
1183 2
1185 2
1187 2
1188 2
1189 2
1190 2
1191 2
1192 2
1193 2
1194 2
1195 2
1196 2
1197 2
1198 2
1199 2
1200 2
1201 2
1202 2
1203 2
1204 3
1205 3
1206 3
1207 3
1208 3
1209 3
1210 3
1211 4
1212 4
1213 4
1214 3
1215 3
1216 3
1217 3
1218 3
1219 3
1221 3
1222 3
1223 3
1224 4
1225 4
1226 3
1227 3
1229 3
1230 4
1231 4
1232 4
1233 3
1235 2
1236 1
1238 1
1239 1

    }

    // Get Unix flavored time
    DateTime *dt=(DateTime *) (pr->getValue());
    time_t backupTime=dt->unixTime();
    // time_t backupTime=(DateTime) (pr->getValue())->unixTime();
    // Get all dates if start and end time are zero
    // otherwise make sure the time is in the selected range
    // Also, get complete backups ONLY if that is what is required
    //
    if (
        ( ( starttime==0) && (endtime==0) ) ||
        ( ( starttime<backupTime) && (endTime>backupTime)
        ) &&
        ( ( flags==BACKUP_SELECTION_FLAG_PARTIAL_OK) ||
        bn->getStateBit(STATE_COMPLETE)
        )
    )
    {
        //
        // Ok, we have found a backup time within the
        // range. Add it here.
        //
        RSTRPC_time_list *newtime=(RSTRPC_time_list *)malloc(
            sizeof(struct RSTRPC_time_list));
        if (!newtime)
            rbe_log_stats(0, "plugin.cc - malloc failure");
        return EP_RB_RECOVER_MALLOC_FAILURE;
    }
    // Increment count of valid time entries
    (*numEntries)++;

    newtime->next=NULL;
    newtime->time=backupTime;
    if (newlist)
    {
        newlist->next=newtime;
    }
    newlist=newtime;

    if (first)
    {
        *timesList=newlist;
        first=0;
    }
}

return E_SUCCESS;
}

```

```

1242 //*****
1243 *
1244 * RSTPL_GetCurrentBackupTime
1245 *
1246 * Function Description:
1247 * Retrieve the time of the backup that the current restore context
1248 * is set to and return it in the preallocated buffer.
1249 *
1250 * Parameters:
1251 * context - (I) Pointer to the restore context
1252 * bkupTime - (O) the time of the backup
1253 *
1254 * Return Codes:
1255 * E_SUCCESS - operation completed successfully
1256 * EP_RB_RECOVER_INVALID - call issued out of sequence
1257 * EP_RB_RECOVER_BAD_ARGS - invalid input argument
1258 * EP_RB_RECOVER_NO_CURR_BACKUP - no valid backup currently
1259 *
1260 *****/
1261
1262 errno_ty RSTPL_GetCurrentBackupTime(restore_context *context,
1263 time_t *backupTime)
1264 {
1265     //
1266     // Now look through current backups and put on list if in
1267     // Range
1268     //
1269     struct restoreContextData *rcd=(struct restoreContextData *){
1270         context->appData);
1271
1272     if(!rcd)
1273     {
1274         return EP_RB_RECOVER_RC_APP_DATA_NULL;
1275     }
1276     BackupNode *currentBackup=rcd->currentBackupNode;
1277     if(!currentBackup)
1278     {
1279         return EP_RB_RECOVER_RN_APP_DATA_NULL;
1280     }
1281
1282     RnPropertyDate *pr=(RnPropertyDate *) (currentBackup->getProperty(
1283     PROPERTY_BACKUP_DATE));
1284     if(!pr)
1285     {
1286         *backupTime=0;
1287         return EP_RB_RECOVER_NO_BACKUPTIME;
1288     }
1289     DateTime *dt=(Datetime *) (pr->getValue());
1290     *backupTime=dt->unixTime();
1291
1292     return E_SUCCESS;
1293 }

```

```

Page 59 of 92                                     RSTPI_SetBackupForTime                                     Thu Jan 03 12:52:58 2008
1298 //*****
1299 * Set Backup For Time
1300 * Function Description:
1301 *   Switch to the backup plane of the specified time, or the most recent
1302 *   that is before the specified time, if an exact match is not possible.
1303 *
1304 *
1305 * Parameters:
1306 *   context (I) - Pointer to the restore context
1307 *   forTime (I) - The time for which the backup is requested
1308 *   flags (I) - Backup constraint flags: e.g., full-only/partial-ok
1309 *
1310 * Return Codes:
1311 *   E_SUCCESS - operation completed successfully
1312 *   EP_RB_RECOVER_XXX - backup plane cannot be found
1313 *
1314 ****
1315 errno_ty RSTPI_SetBackupForTime(restore_context *context,
1316                                     const time_t forTime,
1317                                     RSTRPC_backup_flags_ty flags)
1318
1319
1320 1 {
1321
1322 1   struct restoreContextData *rcd=(struct restoreContextData *)
1323 1
1324 1   context->appdata);
1325 1
1326 1   if(!rcd)
1327 1   {
1328 1     return EP_RB_RECOVER_RC_APP_DATA_NULL;
1329 1
1330 1   WiSetNode *wilist=rcd->currentWiSetNode;
1331 1
1332 1   if(!wilist)
1333 1   {
1334 1     return EP_RB_RECOVER_RN_APP_DATA_NULL;
1335 1
1336 1
1337 1   // Loop through each one on the current list and return the one
1338 1   // which we
1339 1   // are looking for.
1340 1
1341 1   // for(RestoreNode *bn=wilist->getFirstChild();bn;
1342 1   // bn=wilist->getNextChild())
1343 1
1344 1   {
1345 1     RnPropertyDate *pr=(RnPropertyDate *) (bn->getProperty(
1346 1     PROPERTY_BACKUP_DATE));
1347 1
1348 1     if(!pr)
1349 1     {
1350 1       continue;
1351 1     }
1352 1
1353 2   time_t backupTime=dt->uniXtime();
1354 2
1355 2   if ( (forTime >= backupTime) &&
1356 2     ( (
```

1394 //

1395 /*****

1396 *
1397 * Set Previous Backup
1398 *
1399 * Function Description:
1400 * Set the restore context to that of the previous backup with
1401 * respect
1402 * to the current one.
1403 * Parameters:
1404 * context (I) - Pointer to the restore context
1405 * flags (I) - Backup constraint flags: e.g., full-only/partial-ok
1406 *
1407 * Return Codes:
1408 * E_SUCCESS - operation completed successfully
1409 * EP_RB_NO_PREV_CATALOG - when at the first catalog
1410 * EP_RB_RECOVER_PERMISSION_DENIED - when user cannot access the
1411 * file
1412 * of the new catalog
1413 * *****/

1415 **errno_t RSTPI_SetPrevBackup(restore_context *context,
1416 RSTRPC_backup_flags_ty flags)**
1417 {
1418 struct restoreContextData *rcd=(struct restoreContextData *) (context->appData);
1419 struct restoreContextData *rcd=(struct restoreContextData *) (context->appData);
1420 WiSetNode *wilist=rcd->currentWiSetNode;
1421 if(!wilist)
1422 {
1423 1
1424 2
1425 3
1426 4
1427 5
1428 }
1429 5
1430 6
1431 7
1432 8
1433 9
1434 10
1435 11
1436 12
1437 13
1438 14
1439 15
1440 16
1441 17
1442 18
1443 19
1444 20
1445 21
1446 22
1447 23
1448 24
1449 25
1450 26
1451 27
1452 28
1453 29
1454 30
1455 31
1456 32
1457 33
1458 34
1459 35
1460 36
1461 37
1462 38
1463 39
1464 40
1465 41
1466 42
1467 43
1468 44
1469 45
1470 46
1471 47
1472 48
1473 49
1474 50
1475 51
1476 52
1477 53
1478 54
1479 55
1480 56
1481 57
1482 58
1483 59
1484 60
1485 61
1486 62
1487 63
1488 64
1489 65
1490 66
1491 67
1492 68
1493 69
1494 70
1495 71
1496 72
1497 73
1498 74
1499 75
1500 76
1501 77
1502 78
1503 79
1504 80
1505 81
1506 82
1507 83
1508 84
1509 85
1510 86
1511 87
1512 88
1513 89
1514 90
1515 91
1516 92
1517 93
1518 94
1519 95
1520 96
1521 97
1522 98
1523 99
1524 100
1525 101
1526 102
1527 103
1528 104
1529 105
1530 106
1531 107
1532 108
1533 109
1534 110
1535 111
1536 112
1537 113
1538 114
1539 115
1540 116
1541 117
1542 118
1543 119
1544 120
1545 121
1546 122
1547 123
1548 124
1549 125
1550 126
1551 127
1552 128
1553 129
1554 130
1555 131
1556 132
1557 133
1558 134
1559 135
1560 136
1561 137
1562 138
1563 139
1564 140
1565 141
1566 142
1567 143
1568 144
1569 145
1570 146
1571 147
1572 148
1573 149
1574 150
1575 151
1576 152
1577 153
1578 154
1579 155
1580 156
1581 157
1582 158
1583 159
1584 160
1585 161
1586 162
1587 163
1588 164
1589 165
1590 166
1591 167
1592 168
1593 169
1594 170
1595 171
1596 172
1597 173
1598 174
1599 175
1600 176
1601 177
1602 178
1603 179
1604 180
1605 181
1606 182
1607 183
1608 184
1609 185
1610 186
1611 187
1612 188
1613 189
1614 190
1615 191
1616 192
1617 193
1618 194
1619 195
1620 196
1621 197
1622 198
1623 199
1624 200
1625 201
1626 202
1627 203
1628 204
1629 205
1630 206
1631 207
1632 208
1633 209
1634 210
1635 211
1636 212
1637 213
1638 214
1639 215
1640 216
1641 217
1642 218
1643 219
1644 220
1645 221
1646 222
1647 223
1648 224
1649 225
1650 226
1651 227
1652 228
1653 229
1654 230
1655 231
1656 232
1657 233
1658 234
1659 235
1660 236
1661 237
1662 238
1663 239
1664 240
1665 241
1666 242
1667 243
1668 244
1669 245
1670 246
1671 247
1672 248
1673 249
1674 250
1675 251
1676 252
1677 253
1678 254
1679 255
1680 256
1681 257
1682 258
1683 259
1684 260
1685 261
1686 262
1687 263
1688 264
1689 265
1690 266
1691 267
1692 268
1693 269
1694 270
1695 271
1696 272
1697 273
1698 274
1699 275
1700 276
1701 277
1702 278
1703 279
1704 280
1705 281
1706 282
1707 283
1708 284
1709 285
1710 286
1711 287
1712 288
1713 289
1714 290
1715 291
1716 292
1717 293
1718 294
1719 295
1720 296
1721 297
1722 298
1723 299
1724 300
1725 301
1726 302
1727 303
1728 304
1729 305
1730 306
1731 307
1732 308
1733 309
1734 310
1735 311
1736 312
1737 313
1738 314
1739 315
1740 316
1741 317
1742 318
1743 319
1744 320
1745 321
1746 322
1747 323
1748 324
1749 325
1750 326
1751 327
1752 328
1753 329
1754 330
1755 331
1756 332
1757 333
1758 334
1759 335
1760 336
1761 337
1762 338
1763 339
1764 340
1765 341
1766 342
1767 343
1768 344
1769 345
1770 346
1771 347
1772 348
1773 349
1774 350
1775 351
1776 352
1777 353
1778 354
1779 355
1780 356
1781 357
1782 358
1783 359
1784 360
1785 361
1786 362
1787 363
1788 364
1789 365
1790 366
1791 367
1792 368
1793 369
1794 370
1795 371
1796 372
1797 373
1798 374
1799 375
1800 376
1801 377
1802 378
1803 379
1804 380
1805 381
1806 382
1807 383
1808 384
1809 385
1810 386
1811 387
1812 388
1813 389
1814 390
1815 391
1816 392
1817 393
1818 394
1819 395
1820 396
1821 397
1822 398
1823 399
1824 400
1825 401
1826 402
1827 403
1828 404
1829 405
1830 406
1831 407
1832 408
1833 409
1834 410
1835 411
1836 412
1837 413
1838 414
1839 415
1840 416
1841 417
1842 418
1843 419
1844 420
1845 421
1846 422
1847 423
1848 424
1849 425
1850 426
1851 427
1852 428
1853 429
1854 430
1855 431
1856 432
1857 433
1858 434
1859 435
1860 436
1861 437
1862 438
1863 439
1864 440
1865 441
1866 442
1867 443
1868 444
1869 445
1870 446
1871 447
1872 448
1873 449
1874 450
1875 451
1876 452
1877 453
1878 454
1879 455
1880 456
1881 457
1882 458
1883 459
1884 460
1885 461
1886 462
1887 463
1888 464
1889 465
1890 466
1891 467
1892 468
1893 469
1894 470
1895 471
1896 472
1897 473
1898 474
1899 475
1900 476
1901 477
1902 478
1903 479
1904 480
1905 481
1906 482
1907 483
1908 484
1909 485
1910 486
1911 487
1912 488
1913 489
1914 490
1915 491
1916 492
1917 493
1918 494
1919 495
1920 496
1921 497
1922 498
1923 499
1924 500
1925 501
1926 502
1927 503
1928 504
1929 505
1930 506
1931 507
1932 508
1933 509
1934 510
1935 511
1936 512
1937 513
1938 514
1939 515
1940 516
1941 517
1942 518
1943 519
1944 520
1945 521
1946 522
1947 523
1948 524
1949 525
1950 526
1951 527
1952 528
1953 529
1954 530
1955 531
1956 532
1957 533
1958 534
1959 535
1960 536
1961 537
1962 538
1963 539
1964 540
1965 541
1966 542
1967 543
1968 544
1969 545
1970 546
1971 547
1972 548
1973 549
1974 550
1975 551
1976 552
1977 553
1978 554
1979 555
1980 556
1981 557
1982 558
1983 559
1984 560
1985 561
1986 562
1987 563
1988 564
1989 565
1990 566
1991 567
1992 568
1993 569
1994 570
1995 571
1996 572
1997 573
1998 574
1999 575
2000 576
2001 577
2002 578
2003 579
2004 580
2005 581
2006 582
2007 583
2008 584
2009 585
2010 586
2011 587
2012 588
2013 589
2014 590
2015 591
2016 592
2017 593
2018 594
2019 595
2020 596
2021 597
2022 598
2023 599
2024 600
2025 601
2026 602
2027 603
2028 604
2029 605
2030 606
2031 607
2032 608
2033 609
2034 610
2035 611
2036 612
2037 613
2038 614
2039 615
2040 616
2041 617
2042 618
2043 619
2044 620
2045 621
2046 622
2047 623
2048 624
2049 625
2050 626
2051 627
2052 628
2053 629
2054 630
2055 631
2056 632
2057 633
2058 634
2059 635
2060 636
2061 637
2062 638
2063 639
2064 640
2065 641
2066 642
2067 643
2068 644
2069 645
2070 646
2071 647
2072 648
2073 649
2074 650
2075 651
2076 652
2077 653
2078 654
2079 655
2080 656
2081 657
2082 658
2083 659
2084 660
2085 661
2086 662
2087 663
2088 664
2089 665
2090 666
2091 667
2092 668
2093 669
2094 670
2095 671
2096 672
2097 673
2098 674
2099 675
2100 676
2101 677
2102 678
2103 679
2104 680
2105 681
2106 682
2107 683
2108 684
2109 685
2110 686
2111 687
2112 688
2113 689
2114 690
2115 691
2116 692
2117 693
2118 694
2119 695
2120 696
2121 697
2122 698
2123 699
2124 700
2125 701
2126 702
2127 703
2128 704
2129 705
2130 706
2131 707
2132 708
2133 709
2134 710
2135 711
2136 712
2137 713
2138 714
2139 715
2140 716
2141 717
2142 718
2143 719
2144 720
2145 721
2146 722
2147 723
2148 724
2149 725
2150 726
2151 727
2152 728
2153 729
2154 730
2155 731
2156 732
2157 733
2158 734
2159 735
2160 736
2161 737
2162 738
2163 739
2164 740
2165 741
2166 742
2167 743
2168 744
2169 745
2170 746
2171 747
2172 748
2173 749
2174 750
2175 751
2176 752
2177 753
2178 754
2179 755
2180 756
2181 757
2182 758
2183 759
2184 760
2185 761
2186 762
2187 763
2188 764
2189 765
2190 766
2191 767
2192 768
2193 769
2194 770
2195 771
2196 772
2197 773
2198 774
2199 775
2200 776
2201 777
2202 778
2203 779
2204 780
2205 781
2206 782
2207 783
2208 784
2209 785
2210 786
2211 787
2212 788
2213 789
2214 790
2215 791
2216 792
2217 793
2218 794
2219 795
2220 796
2221 797
2222 798
2223 799
2224 800
2225 801
2226 802
2227 803
2228 804
2229 805
2230 806
2231 807
2232 808
2233 809
2234 810
2235 811
2236 812
2237 813
2238 814
2239 815
2240 816
2241 817
2242 818
2243 819
2244 820
2245 821
2246 822
2247 823
2248 824
2249 825
2250 826
2251 827
2252 828
2253 829
2254 830
2255 831
2256 832
2257 833
2258 834
2259 835
2260 836
2261 837
2262 838
2263 839
2264 840
2265 841
2266 842
2267 843
2268 844
2269 845
2270 846
2271 847
2272 848
2273 849
2274 850
2275 851
2276 852
2277 853
2278 854
2279 855
2280 856
2281 857
2282 858
2283 859
2284 860
2285 861
2286 862
2287 863
2288 864
2289 865
2290 866
2291 867
2292 868
2293 869
2294 870
2295 871
2296 872
2297 873
2298 874
2299 875
2300 876
2301 877
2302 878
2303 879
2304 880
2305 881
2306 882
2307 883
2308 884
2309 885
2310 886
2311 887
2312 888
2313 889
2314 890
2315 891
2316 892
2317 893
2318 894
2319 895
2320 896
2321 89


```

1488 // ****
1489 *
1490 * Set Next Backup
1491
1492 * Function Description:
1493 * This routine must set the restore environment to the the next
1494 * of the current top level object.
1495 *
1496 * Parameters:
1497 *   context (I) - Pointer to the restore context
1498 *   flags (I) - Backup constraint flags: e.g., full-only/partial-ok
1499 *
1500 * Return Codes:
1501 *   E_SUCCESS - operation completed successfully
1502 *   EP_RB_RECOVER_NO_NEXT_CATALOG - when at the most recent
1503 *   EP_RB_RECOVER_PERMISSION_DENIED - when user cannot access the
1504 *   catalog of the new catalog
1505 *   EP_RB_RECOVER_NO_CATALOG - when meat_set_moplane failed
1506 *   EP_RB_RECOVER_WI_LOCKED - when we found the one to backup
1507 *   EP_RB_RECOVER_NONE_EXISTS;
1508 *   EP_RB_RECOVER_WI_LOCKED;
1509 *   RSTPI_SetNextBackup (restore_context *context,
1510 *   RSTRPC_backup_flags_ty flags)
1511 {
1512
1513     struct restoreContextData *rcd=(struct restoreContextData *)(
1514         context->appData);
1515
1516     WiSetNode *wiList=rqd->currentWiSetNode;
1517
1518     if(!wiList)
1519     {
1520         return EP_RB_RECOVER_RC_APP_DATA_NULL;
1521     }
1522
1523     1
1524     BackupNode *bnode=rqd->currentBackupNode;
1525
1526     1
1527     if(bnode)
1528     {
1529         1
1530         {
1531             1
1532             1
1533             1
1534             1
1535             1
1536             1
1537             2
1538             2
1539             3
1540             3
1541             2
1542             2
1543             1
1544             1
1545             1
1546             1
1547             2
1548             2
1549             2
1550             3
1551             3
1552             3
1553             2
1554             2
1555             3
1556             3
1557             2
1558             2
1559             2
1560             2
1561             3
1562             3
1563             2
1564             2
1565             2
1566             1
1567             2
1568             1
1569             1
1570             1
1571             1
1572             1
1573             1
1574             1
1575             1
1576             1
1577             1
1578             1
1579             1
1580             1
1581             1
1582             1
1583             1
1584             1
1585             1
1586             1
1587             1
1588             1
1589             1
1590             1
1591             1
1592             1
1593             1
1594             1
1595             1
1596             1
1597             1
1598             1
1599             1
1600             1
1601             1
1602             1
1603             1
1604             1
1605             1
1606             1
1607             1
1608             1
1609             1
1610             1
1611             1
1612             1
1613             1
1614             1
1615             1
1616             1
1617             1
1618             1
1619             2
1620             2
1621             1
1622             1
1623             1
1624             1
1625             1
1626             1
1627             1
1628             1
1629             1
1630             1
1631             1
1632             1
1633             1
1634             1
1635             1
1636             1
1637             2
1638             2
1639             3
1640             3
1641             2
1642             2
1643             1
1644             1
1645             1
1646             1
1647             1
1648             1
1649             1
1650             1
1651             1
1652             1
1653             1
1654             1
1655             1
1656             1
1657             1
1658             1
1659             1
1660             1
1661             1
1662             1
1663             1
1664             1
1665             1
1666             1
1667             1
1668             1
1669             1
1670             1
1671             1
1672             1
1673             1
1674             1
1675             1
1676             1
1677             1
1678             1
1679             1
1680             1
1681             1
1682             1
1683             1
1684             1
1685             1
1686             1
1687             1
1688             1
1689             1
1690             1
1691             1
1692             1
1693             1
1694             1
1695             1
1696             1
1697             1
1698             1
1699             1
1700             1
1701             1
1702             1
1703             1
1704             1
1705             1
1706             1
1707             1
1708             1
1709             1
1710             1
1711             1
1712             1
1713             1
1714             1
1715             1
1716             1
1717             1
1718             1
1719             1
1720             1
1721             1
1722             1
1723             1
1724             1
1725             1
1726             1
1727             1
1728             1
1729             1
1730             1
1731             1
1732             1
1733             1
1734             1
1735             1
1736             1
1737             1
1738             1
1739             1
1740             1
1741             1
1742             1
1743             1
1744             1
1745             1
1746             1
1747             1
1748             1
1749             1
1750             1
1751             1
1752             1
1753             1
1754             1
1755             1
1756             1
1757             1
1758             1
1759             1
1760             1
1761             1
1762             1
1763             1
1764             1
1765             1
1766             1
1767             1
1768             1
1769             1
1770             1
1771             1
1772             1
1773             1
1774             1
1775             1
1776             1
1777             1
1778             1
1779             1
1780             1
1781             1
1782             1
1783             1
1784             1
1785             1
1786             1
1787             1
1788             1
1789             1
1790             1
1791             1
1792             1
1793             1
1794             1
1795             1
1796             1
1797             1
1798             1
1799             1
1800             1
1801             1
1802             1
1803             1
1804             1
1805             1
1806             1
1807             1
1808             1
1809             1
1810             1
1811             1
1812             1
1813             1
1814             1
1815             1
1816             1
1817             1
1818             1
1819             1
1820             1
1821             1
1822             1
1823             1
1824             1
1825             1
1826             1
1827             1
1828             1
1829             1
1830             1
1831             1
1832             1
1833             1
1834             1
1835             1
1836             1
1837             1
1838             1
1839             1
1840             1
1841             1
1842             1
1843             1
1844             1
1845             1
1846             1
1847             1
1848             1
1849             1
1850             1
1851             1
1852             1
1853             1
1854             1
1855             1
1856             1
1857             1
1858             1
1859             1
1860             1
1861             1
1862             1
1863             1
1864             1
1865             1
1866             1
1867             1
1868             1
1869             1
1870             1
1871             1
1872             1
1873             1
1874             1
1875             1
1876             1
1877             1
1878             1
1879             1
1880             1
1881             1
1882             1
1883             1
1884             1
1885             1
1886             1
1887             1
1888             1
1889             1
1890             1
1891             1
1892             1
1893             1
1894             1
1895             1
1896             1
1897             1
1898             1
1899             1
1900             1
1901             1
1902             1
1903             1
1904             1
1905             1
1906             1
1907             1
1908             1
1909             1
1910             1
1911             1
1912             1
1913             1
1914             1
1915             1
1916             1
1917             1
1918             1
1919             1
1920             1
1921             1
1922             1
1923             1
1924             1
1925             1
1926             1
1927             1
1928             1
1929             1
1930             1
1931             1
1932             1
1933             1
1934             1
1935             1
1936             1
1937             1
1938             1
1939             1
1940             1
1941             1
1942             1
1943             1
1944             1
1945             1
1946             1
1947             1
1948             1
1949             1
1950             1
1951             1
1952             1
1953             1
1954             1
1955             1
1956             1
1957             1
1958             1
1959             1
1960             1
1961             1
1962             1
1963             1
1964             1
1965             1
1966             1
1967             1
1968             1
1969             1
1970             1
1971             1
1972             1
1973             1
1974             1
1975             1
1976             1
1977             1
1978             1
1979             1
1980             1
1981             1
1982             1
1983             1
1984             1
1985             1
1986             1
1987             1
1988             1
1989             1
1990             1
1991             1
1992             1
1993             1
1994             1
1995             1
1996             1
1997             1
1998             1
1999             1
2000             1
2001             1
2002             1
2003             1
2004             1
2005             1
2006             1
2007             1
2008             1
2009             1
2010             1
2011             1
2012             1
2013             1
2014             1
2015             1
2016             1
2017             1
2018             1
2019             1
2020             1
2021             1
2022             1
2023             1
2024             1
2025             1
2026             1
2027             1
2028             1
2029             1
2030             1
2031             1
2032             1
2033             1
2034             1
2035             1
2036             1
2037             1
2038             1
2039             1
2040             1
2041             1
2042             1
2043             1
2044             1
2045             1
2046             1
2047             1
2048             1
2049             1
2050             1
2051             1
2052             1
2053             1
2054             1
2055             1
2056             1
2057             1
2058             1
2059             1
2060             1
2061             1
2062             1
2063             1
2064             1
2065             1
2066             1
2067             1
2068             1
2069             1
2070             1
2071             1
2072             1
2073             1
2074             1
2075             1
2076             1
2077             1
2078             1
2079             1
2080             1
2081             1
2082             1
2083             1
2084             1
2085             1
2086             1
2087             1
2088             1
2089             1
2090             1
2091             1
2092             1
2093             1
2094             1
2095             1
2096             1
2097             1
2098             1
2099             1
2100             1
2101             1
2102             1
2103             1
2104             1
2105             1
2106             1
2107             1
2108             1
2109             1
2110             1
2111             1
2112             1
2113             1
2114             1
2115             1
2116             1
2117             1
2118             1
2119             1
2120             1
2121             1
2122             1
2123             1
2124             1
2125             1
2126             1
2127             1
2128             1
2129             1
2130             1
2131             1
2132             1
2133             1
2134             1
2135             1
2136             1
2137             1
2138             1
2139             1
2140             1
2141             1
2142             1
2143             1
2144             1
2145             1
2146             1
2147             1
2148             1
2149             1
2150             1
2151             1
2152             1
2153             1
2154             1
2155             1
2156             1
2157             1
2158             1
2159             1
2160             1
2161             1
2162             1
2163             1
2164             1
2165             1
2166             1
2167             1
2168             1
2169             1
2170             1
2171             1
2172             1
2173             1
2174             1
2175             1
2176             1
2177             1
2178             1
2179             1
2180             1
2181             1
2182             1
2183             1
2184             1
2185             1
2186             1
2187             1
2188             1
2189             1
2190             1
2191             1
2192             1
2193             1
2194             1
2195             1
2196             1
2197             1
2198             1
2199             1
2200             1
2201             1
2202             1
2203             1
2204             1
2205             1
2206             1
2207             1
2208             1
2209             1
2210             1
2211             1
2212             1
2213             1
2214             1
2215             1
2216             1
2217             1
2218             1
2219             1
2220             1
2221             1
2222             1
2223             1
2224             1
2225             1
2226             1
2227             1
2228             1
2229             1
2230             1
2231             1
2232             1
2233             1
2234             1
2235             1
2236             1
2237             1
2238             1
2239             1
2240             1
2241             1
2242             1
2243             1
2244             1
2245             1
2246             1
2247             1
2248             1
2249             1
2250             1
2251             1
2252             1
2253             1
2254             1
2255             1
2256             1
2257             1
2258             1
2259             1
2260             1
2261             1
2262             1
2263             1
2264             1
2265             1
2266             1
2267             1
2268             1
2269             1
2270             1
2271             1
2272             1
2273             1
2274             1
2275             1
2276             1
2277             1
2278             1
2279             1
2280             1
2281             1
2282             1
2283             1
2284             1
2285             1
2286             1
2287             1
2288             1
2289             1
2290             1
2291             1
2292             1
2293             1
2294             1
2295             1
2296             1
2297             1
2298             1
2299             1
2300             1
2301             1
2302             1
2303             1
2304             1
2305             1
2306             1
2307             1
2308             1
2309             1
2310             1
2311             1
2312             1
2313             1
2314             1
2315             1
2316             1
2317             1
2318             1
2319             1
2320             1
2321             1
2322             1
2323             1
2324             1
2325             1
2326             1
2327             1
2328             1
2329             1
2330             1
2331             1
2332             1
2333             1
2334             1
2335             1
2336             1
2337             1
2338             1
2339             1
2340             1
2341             1
2342             1
2343             1
2344             1
2345             1
2346             1
2347             1
2348             1
2349             1
2350             1
2351             1
2352             1
2353             1
2354             1
2355             1
2356             1
2357             1
2358             1
2359             1
2360             1
2361             1
2362             1
2363             1
2364             1
2365             1
2366             1
2367             1
2368             1
2369             1
2370             1
2371             1
2372             1
2373             1
2374             1
2375             1
2376             1
2377             1
2378             1
2379             1
2380             1
2381             1
2382             1
2383             1
2384             1
2385             1
2386             1
2387             1
2388             1
2389             1
2390             1
2391             1
2392             1
2393             1
2394             1
2395             1
2396             1
2397             1
2398             1
2399             1
2400             1
2401             1
2402             1
2403             1
2404             1
2405             1
2406             1
2407             1
2408             1
2409             1
2410             1
2411             1
2412             1
2413             1
2414             1
2415             1
2416             1
2417             1
2418             1
2419             1
2420             1
2421             1
2422             1
2423             1
2424             1
2425             1
2426             1
2427             1
2428             1
2429             1
2430             1
2431             1
2432             1
2433             1
2434             1
2435             1
2436             1
2437             1
2438             1
2439             1
2440             1
2441             1
2442             1
2443             1
2444             1
2445             1
2446             1
2447             1
2448             1
2449             1
2450             1
2451             1
2452             1
2453             1
2454             1
2455             1
2456             1
2457             1
2458             1
2459             1
2460             1
2461             1
2462             1
2463             1
2464             1
2465             1
2466             1
2467             1
2468             1
2469             1
2470             1
2471             1
2472             1
2473             1
2474             1
2475             1
2476             1
2477             1
2478             1
2479             1
2480             1
2481             1
2482             1
2483             1
2484             1
2485             1
2486             1
2487             1
2488             1
2489             1
2490             1
2491             1
2492             1
2493             1
2494             1
2495             1
2496             1
2497             1
2498             1
2499             1
2500             1
2501             1
2502             1
2503             1
2504             1
2505             1
2506             1
2507             1
2508             1
2509             1
2510             1
2511             1
2512             1
2513             1
2514             1
2515             1
2516             1
2517             1
2518             1
2519             1
2520             1
2521             1
2522             1
2523             1
2524             1
2525             1
2526             1
2527             1
2528             1
2529             1
2530             1
2531             1
2532             1
2533             1
2534             1
2535             1
2536             1
2537             1
2538             1
2539             1
2540             1
2541             1
2542             1
2543             1
2544             1
2545             1
2546             1
2547             1
2548             1
2549             1
2550             1
2551             1
2552             1
2553             1
2554             1
2555             1
2556             1
2557             1
2558             1
2559             1
2560             1
2561             1
2562             1
2563             1
2564             1
2565             1
2566             1
2567             1
2568             1
2569             1
2570             1
2571             1
2572             1
2573             1
2574             1
2575             1
2576             1
2577             1
2578             1
2579             1
2580             1
2581             1
2582             1
2583             1
2584             1
2585             1
2586             1
2587             1
2588             1
2589             1
2590             1
2591             1
2592             1
2593             1
2594             1
2595             1
2596             1
2597             1
2598             1
2599             1
2600             1
2601             1
2602             1
2603             1
2604             1
2605             1
2606             1
2607             1
2608             1
2609             1
2610             1
2611             1
2612             1
2613             1
2614             1
2615             1
2616             1
2617             1
2618             1
2619             1
2620             1
2621             1
2622             1
2623             1
2624             1
2625             1
2626             1
2627             1
2628             1
2629             1
2630             1
2631             1
2632             1
2633             1
2634             1
2635             1
2636             1
2637             1
2638             1
2639             1
2640             1
2641             1
2642             1
2643             1
2644             1
2645             1
2646             1
2647             1
2648             1
2649             1
2650             1
2651             1
2652             1
2653             1
2654             1
2655             1
2656             1
2657             1
2658             1
2659             1
2660             1
2661             1
2662             1
2663             1
2664             1
2665             1
2666             1
2667             1
2668             1
2669             1
2670             1
2671             1
2672             1
2673             1
2674             1
2675             1
2676             1
2677             1
2678             1
2679             1
2680             1
2681             1
2682             1
2683             1
2684             1
2685             1
2686             1
2687             1
2688             1
2689             1
2690             1
2691             1
2692             1
2693             1
2694             1
2695             1
2696             1
2697             1
2698             1
2699             1
2700             1
2701             1
2702             1
2703             1
2704             1
2705             1
2706             1
2707             1
2708             1
2709             1
2710             1
2711             1
2712             1
2713             1
2714             1
2715             1
2716             1
2717             1
2718             1
2719             1
2720             1
2721             1
2722             1
2723             1
2724             1
2725             1
2726             1
2727             1
2728             1
2729             1
2730             1
2731             1
2732             1
2733             1
2734             1
2735             1
2736             1
2737             1
2738             1
2739             1
2740             1
2741             1
2742             1
2743             1
2744             1
2745             1
2746             1
2747             1
2748             1
2749             1
2750             1
2751             1
2752             1
2753             1
2754             1
2755             1
2756             1
2757             1
2758             1
2759             1
2760             1
2761             1
2762             1
2763             1
2764             1
2765             1
2766             1
2767             1
2768             1
2769             1
2770             1
2771             1
2772             1
2773             1
2774             1
2775             1
2776             1
2777             1
2778             1
2779             1
2780             1
2781             1
2782             1
2783             1
2784             1
2785             1
2786             1
2787             1
2788             1
2789             1
2790             1
2791             1
2792             1
2793             1
2794             1
2795             1
2796             1
2797             1
2798             1
2799             1
2800             1
2801             1
2802             1
2803             1
2804             1
2805             1
2806             1
2807             1
2808             1
2809             1
2810             1
2811             1
2812             1
2813             1
2814             1
2815             1
2816             1
2817             1
2818             1
2819             1
2820             1
2821             1
2822             1
2823             1
2824             1
2825             1
2826             1
2827             1
2828             1
2829             1
2830             1
2831             1
2832             1
2833             1
2834             1
2835             1
2836             1
2837             1
2838             1
2839             1
2840             1
2841             1
2842             1
2843             1
2844             1
2845             1
2846             1
2847             1
2848             1
2849             1
2850             1
2851             1
2852             1
2853             1
2854             1
2855             1
2856             1
2857             1
2858             1
2859             1
2
```

```

1579 // *****
1580 * Set Most Recent Backup
1581 * Set Most Recent Backup
1582 * Function Description:
1583 * Set the restore context to that of the most recent backup catalog
1584 * plane.
1585 *
1586 * Parameters:
1587 * context (I) - Pointer to the restore context
1588 * flags (
1589 * I) - Backup constraint flags: e.g., full-only/partial-ok
1590 * Return Codes:
1591 * E_SUCCESS - operation completed successfully
1592 * EP_RB_RECOVER_PERMISSION_DENIED - when user cannot access the
1593 * file of the new catalog.
1594 *
1595 * *****
1596 * *****
1597 errno_ty RSTPI_SetMostRecentBackup(RSTRPC_backup_flags_ty flags)
1599 1600 {
1601 1 {
1603 1 struct restoreContextData *rcd=(struct restoreContextData *)(
1604 1 context->appData);
1606 1 WiSetNode *wilist=rcd->currentWiSetNode;
1608 1 if(!wilist)
1609 2 {
1610 2   return EP_RB_RECOVER_RC_APP_DATA_NULL;
1611 1 }
1613 1
1614 1 RestoreNode *bnode=NULL;
1615 1
1616 1 // Set the iterator to the current item
1617 1
1618 1 for(RestoreNode *rn=wilist->getFirstChild();rn;
1619 1 rn=wilist->getNextChild())
1620 2
1621 2   if( flags==BACKUP_SELECTION_FLAG_PARTIAL_OK || rn->getStateBit(
1622 3 STATE_COMPLETE))
1623 3
1624 3   {
1625 4     // If we already have a current backup node, clear all marks
1626 4     if(rcd->currentBackupNode)
1627 3       if(NULL != rcd->currentBackupNode)
1628 3         {
1629 4           if(rn==rcd->currentBackupNode)
1630 5             {
1631 5               /* We are already at the right place so return
1632 5               success */
1633 5             }
1634 4           rcd->currentBackupNode->unlockWorkItems(context);
1635 4

```

```
1636 3
1637 3
1638 3
1639 3
1640 4
    }
    BackupNode *bnode=(BackupNode *)rn;
    rcd->currentBackupNode=bnode;
    if(!rcd->currentBackupNode->lockWorkItems(context))
    {
        return EP_RB_RECOVER_WI_LOCKED;
    }
    bnode->populate(context,POPULATE_CHILDREN);
    // We found the one to backup
    return E_SUCCESS;
}
1646 3
1647 2
1648 1
}
return EP_RB_RECOVER_SET_BUTIME_ERROR;
1651 1
1652 1
}
```

```

1659 //*****
1660 * Set First Backup
1661 * Function Description:
1662 * Set the restore context to that of the first backup plane.
1663 *
1664 * Parameters:
1665 *   context (I) - Pointer to the restore context
1666 *   flags (I) - Backup constraint flags: e.g., full-only/partial-ok
1667 *   Return Codes:
1668 *     E_SUCCESS - operation completed successfully
1669 *     EP_RB_RECOVER_PERMISSION_DENIED - when user cannot access the
1670 *                                         file of
1671 *                                         the new catalog
1672 *                                         *****/
1673 *
1674 * *****/
1675 *
1676 *
1677 *
1678 *errno_ty RSTPI_SetFirstBackup	restore_context *context,
1679 *RSTRPC_Backup_flags_ty flags)
1680 *
1681 {
1682     struct restoreContextData *rcd=(struct restoreContextData *)(
1683         context->appData);
1684
1685     WiSetNode *wlist=rcd->currentWiSetNode;
1686
1687     if(!wlist)
1688     {
1689         return EP_RB_RECOVER_RC_APP_DATA_NULL;
1690     }
1691
1692     // set the iterator to the current item
1693     // set the iterator to the current item
1694
1695     for(RestoreNode *rn=wlist->getFirstChild();rn;
1696         rn=wlist->getNextChild())
1697     {
1698         if( flags==BACKUP_SELECTION_FLAG_PARTIAL_OK || rn->getStateBit(
1699             STATE_COMPLETE)
1700             {
1701                 // If we already have a current backup node, clear all marks
1702                 if(rcd->currentBackupNode)
1703                 {
1704                     rcd->currentBackupNode->unmarkNode(TRUE, FALSE);
1705                     if(NULL!=rcd->currentBackupNode)
1706                     {
1707                         rcd->currentBackupNode->unlockWorkItems(context);
1708                     }
1709                     BackupNode *bnode=(BackupNode *)rn;
1710                     rcd->currentBackupNode=bnode;
1711                     if(rcd->currentBackupNode->lockWorkItems(context))
1712                     {
1713                         return EP_RB_RECOVER_WL_LOCKED;
1714                     }
1715                     bnode->populate(context, POPULATE_CHILDREN);
1716
1717     }
1718 }
1719 }
1720
1721 return EP_RB_RECOVER_SET_BUTIME_ERROR;
1722 }

```

```

1730 // ****
1731 * Is There Prev Backup
1732 *
1733 * Function Description:
1734 * Determine if a backup exists prior to the backup that is
1735 * currently selected.
1736 *
1737 * Parameters:
1738 * context (I) - Pointer to the restore context
1739 * flags (
1740 * I) - Backup constraint flags: e.g., full-only/partial-ok
1741 * isThere (O) - TRUE/FALSE that requested backup does exist
1742 *
1743 * Return Codes:
1744 * E_SUCCESS
1745 * EP_RB_RECOVER_xxx
1746 * - operation completed successfully
1747 * - when errors occur accessing catalogs
1748 * ****
1749 errno_ty RSTPL_IsTherePrevBackup (restore_context *context,
1750 RSTRPC_backup_flags_ty flags,
1751 boolean_ty *isThere)
1752 {
1753     struct restoreContextData *rccd=(struct restoreContextData *)(
1754         context->appData);
1755     *isThere=FALSE;
1756     WiSetNode *wilist=rccd->currentWiSetNode;
1757     if(!wilist)
1758     {
1759         if(!wilist)
1760         {
1761             return EP_RB_RECOVER_RC_APP_DATA_NULL;
1762         }
1763     }
1764     BackupNode *bnode=rccd->currentBackupNode;
1765     if(bnode)
1766     {
1767         //Debugging only
1768         RestoreNode *rob=bnode;
1769         if(!bnode)
1770         {
1771             if(!wilist)
1772             {
1773                 return EP_RB_RECOVER_RC_APP_DATA_NULL;
1774             }
1775             // Set the iterator to the current item
1776             if(!wilist->setIterToSpecificChild(bnode))
1777             {
1778                 if(first)
1779                 {
1780                     if(first)
1781                     {
1782                         for(RestoreNode *rn=wilist->getThisChild();rn=
1783                             rn=wilist->getNextChild())
1784                         if(first)
1785                         {
1786                             if(first)
1787                             {
1788                                 if(first)
1789                                 {
1790

```

```

1789 3     {
1790 3         first=FALSE;
1791 2     }
1792 2     else
1793 3     {
1794 3         if(
1795 4             flags==BACKUP_SELECTION_FLAG_PARTIAL_OK || rn->getStateBit(
1796 4                 STATE_COMPLETE)
1797 4             *isThere==TRUE;
1798 3         )
1799 2     }
1800 1 }
1801 1     return E_SUCCESS;
1802 1 }
1803
}

```

```

1808 // *****
1809 * Is There Next Backup
1810 *
1811 * Function Description:
1812 * Determine if a backup exists after the backup that is
1813 * currently selected.
1814 *
1815 * Parameters:
1816 * context (I) - Pointer to the restore context
1817 * flags (
1818 * I) - Backup constraint flags: e.g., full-only/partial-ok
1819 * isThere (O) - TRUE/FALSE that requested backup does exist
1820 *
1821 * Return Codes:
1822 * E_SUCCESS - operation completed successfully
1823 * EP_RB_RECOVER_XXX - when errors occur accessing catalogs
1824 *
1825 *****/
1826
1827
1828 RSTPI_IsThereNextBackup	restore_context *context,
1829 RSRPC_backup_flags_ty flags,
1830 boolean_ty *isThere)
1831 {
1832 struct restoreContextData *rcdd=(struct restoreContextData *)(
1833 context->appData);
1834
1835 *isThere=FALSE;
1836
1837 WiSetNode *wilist=rcd->currentWiSetNode;
1838
1839 if(!wilist)
1840 {
1841
1842 return EP_RB_RECOVER_RC_APP_DATA_NULL;
1843
1844
1845 BackupNode *bnode=rcd->currentBackupNode;
1846
1847 if(!bnode)
1848 {
1849
1850 rn=rnode->getFirstChild();
1851
1852
1853 while(rn && rn != bnode)
1854
1855 {
1856 if( flags==BACKUP_SELECTION_FLAG_PARTIAL_OK || rn->getStateBit(
1857 STATE_COMPLETE))
1858
1859
1860 rn=rnode->getNextChild();
1861
1862
1863
1864 return E_SUCCESS;
1865
1866 }

```

```

1869 // *****
1870 * Is Object Markable
1871 *
1872 * Function Description:
1873 * Returns TRUE if the specified object is markable by the user,
1874 * returns FALSE if it is not. This function applies only to
1875 * container (directory) and leaf (file) objects.
1876 *
1877 * Parameters:
1878 * context - (I) Pointer to the restore context
1879 * thisObject - (I) ptr to the restorableObject in question
1880 *
1881 * Return:
1882 * TRUE - the specified object is markable by the user;
1883 * FALSE - the specified object is not markable by the user;
1884 *
1885 *****/
1886
1887 RSTPI_IsObjectMarkable(
1888
1889 restore_context
1890 struct RSRPC_user_restorable_object
1891 *thisObject
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3398
3399
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3498
3499
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3598
3599
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3698
3699
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3789
3790
37
```

```

1898
1899 // *****
1900 * Get Top Level Templates:
1901 * This function is required to retrieve the templates with which a
1902 * object could have been backed up.
1903 *
1904 * Parameters:
1905 *   context          (I) - Pointer to the restore context
1906 *   toplevelobj     (I) - the top level object
1907 *   templates        (
1908 *     O) - pointer to receive the start of the list of templates
1909 *   numberEntries   (O) - the real number of templates returned in the list
1910 * *****
1912 errno_ty RSTPI_GetTopLevelTemplates(
1913     restore_context           *context,
1914     struct RSTRPC_top_level_obj *toplevelobj,
1915     struct RSTRPC_name_list   **templates,
1916     short                      *numberEntries )
1917 {
1918     char ** c1=(char **)(toplevelobj->appData.data);
1919     if(NULL==rnodep)
1920     {
1921         rbe_log_stats(
1922             0, "RSTPI_GetTopLevelTemplates - Mark object has no app data");
1923         return EP_RB_RECOVER_RN_APP_DATA_NULL;
1924     }
1925     // Since we are setting a top level object, the marks on any existing
1926     // object must be cleared.
1927     // struct restoreContextData *rcd=(struct restoreContextData *)
1928     // if(NULL!=rcd)
1929     // {
1930         struct restoreContextData *rcd=(struct restoreContextData *)
1931         if(NULL!=rcd)
1932         {
1933             if(NULL!=rcd->currentBackupNode)
1934             {
1935                 rcd->currentBackupNode->unmarkNode(TRUE, FALSE);
1936             }
1937         }
1938         *numberEntries=NULL;
1939         *templates=NULL;
1940     }
1941
1942     for(RestoreNode *rn=rnodep->getFirstChild();
1943         rn=rnodep->getNextChild())
1944     {
1945         if(rn->nodeType() != RNC_BACKUP)
1946         {
1947             continue;
1948         }
1949         if(rn->nodeType() != RNC_BACKUP)
1950         {
1951             continue;
1952         }
1953
1954     }
1955     if(NULL==tp)
1956     {
1957         0, "RSTPI_GetTopLevelTemplates: Template property not found on backup
1958         node");
1959         }
1960         continue;
1961     }
1962         RnPropertyChar *cp=(RnPropertyChar *)tp;
1963         char * tname=(char *) (cp->getValue());
1964         // We should always find a template property. Skip this item
1965         // if we don't.
1966         //
1967         if(tname[0] == '\0')
1968         {
1969             rbe_log_stats(
1970                 0, "RSTPI_GetTopLevelTemplates: Template property ob backup node has no
1971                 value");
1972             continue;
1973         }
1974         // Now, let's see if we already have it...
1975         if(lookat_entry)
1976         {
1977             bool addIt=TRUE;
1978             struct RSTRPC_name_list *lookat_next;
1979             while(lookat_entry)
1980             {
1981                 struct RSTRPC_name_list *lookat_entry=templates;
1982                 if(0==strcmp(tname, lookat_entry->name) )
1983                 {
1984                     lookat_next=lookat_entry->next;
1985                     if(0==strcmp(tname, lookat_entry->name) )
1986                     {
1987                         lookat_entry->next=NULL;
1988                         addIt=FALSE; // Found it!!!
1989                     }
1990                     lookat_entry=lookat_next;
1991                 }
1992                 // Add the entry only if it hasn't already been placed on the
1993                 // list
1994                 if(addIt)
1995                 {
1996                     // Add entry to the list if not already here...
1997                     struct RSTRPC_name_list *entry;
1998                     entry=(struct RSTRPC_name_list *)malloc(sizeof(
1999                         struct RSTRPC_name_list));
2000                     if(NULL==entry)
2001                     {
2002                         rbe_log_stats(
2003                             0, "RSTPI_GetTopLevelTemplates: Malloc failure");
2004                         return EP_RB_RECOVER_MALLOC_FAILURE;
2005                     }
2006                     entry->next=*templates;
2007                     entry->name=tname);
2008                     entry->name=es1_strdup(tname);
2009                 }
2010             }
2011         }
2012     }

```

```

1954     RnProperty *tp=rn->getProperty(PROPERTY_TEMPLATE);
1955     if(NULL==tp)
1956     {
1957         rbe_log_stats(
1958             0, "RSTPI_GetTopLevelTemplates: Template property not found on backup
1959             node");
1960         continue;
1961     }
1962         RnPropertyChar *cp=(RnPropertyChar *)tp;
1963         char * tname=(char *) (cp->getValue());
1964         // We should always find a template property. Skip this item
1965         // if we don't.
1966         //
1967         if(tname[0] == '\0')
1968         {
1969             rbe_log_stats(
1970                 0, "RSTPI_GetTopLevelTemplates: Template property ob backup node has no
1971                 value");
1972             continue;
1973         }
1974         // Now, let's see if we already have it...
1975         if(lookat_entry)
1976         {
1977             bool addIt=TRUE;
1978             struct RSTRPC_name_list *lookat_next;
1979             while(lookat_entry)
1980             {
1981                 struct RSTRPC_name_list *lookat_entry=templates;
1982                 if(0==strcmp(tname, lookat_entry->name) )
1983                 {
1984                     lookat_next=lookat_entry->next;
1985                     if(0==strcmp(tname, lookat_entry->name) )
1986                     {
1987                         lookat_entry->next=NULL;
1988                         addIt=FALSE; // Found it!!!
1989                     }
1990                     lookat_entry=lookat_next;
1991                 }
1992                 // Add the entry only if it hasn't already been placed on the
1993                 // list
1994                 if(addIt)
1995                 {
1996                     // Add entry to the list if not already here...
1997                     struct RSTRPC_name_list *entry;
1998                     entry=(struct RSTRPC_name_list *)malloc(sizeof(
1999                         struct RSTRPC_name_list));
2000                     if(NULL==entry)
2001                     {
2002                         rbe_log_stats(
2003                             0, "RSTPI_GetTopLevelTemplates: Malloc failure");
2004                         return EP_RB_RECOVER_MALLOC_FAILURE;
2005                     }
2006                     entry->next=*templates;
2007                     entry->name=tname);
2008                     entry->name=es1_strdup(tname);
2009                 }
2010             }
2011         }
2012     }

```

```

2013 3
2014 4
2015 4
2016 4
2017 3
2018 3
2019 3
2020 3
2021 3
2022 2
2023 1
2024 1
2025 1
2026 1
2027 1
2028 1
2029 1
2030 1
2031 1
2032 1
2033 1
2034 1
2035 1
2036 1
2037 1
2038 1
2039 1
2040 1
2041 1
2042 1
2043 1
2044 1
2045 1
2046 1
2047 1
2048 1
2049 1
2050 1
2051 1
2052 1
2053 1
2054 1
2055 1
2056 1
2057 2
2058 2
2059 1
2060 1
2061 1
2062 1
2063 2
2064 2
2065 2
2066 2
2067 2
2068 1
2069 1
2070 1
2071 1
2072 1

    if(NULL == entry->name)
    {
        rbe_log_stats(
            0, "RSTPI_GetTopLevelTemplates: Malloc failure");
        return EP_RB_RECOVER_MALLOC_FAILURE;
    }

    *templates=entry;
    (*numberEntries)++;
}

Returns:
    * E_SUCCESS
    * EP_RB_RECOVER_xxx on success
    * EP_RB on error

Parameters:
    * context
        (I) - Pointer to the restore context
    * restore_context
        (I) - Pointer to the restore context
    * struct restoreContextData *rcd=(struct restoreContextData *)
        context->appdata

rbe_log_debug(0, "In RSTPI_ClearRestoreContext");

if(NULL == rcd)
{
    return EP_RB_RECOVER_RC_APP_DATA_NULL;
}

// Clear locks if they exist
if(NULL != rcd->currentBackupNode)
{
    rbe_log_debug(0, "RSTPI_ClearRestoreContext: unlocking %s",
        rcd->currentBackupNode->unlockWorkItems(context));
    rcd->currentBackupNode->unmarkNode(TRUE, FALSE);
    rcd->currentBackupNode=NULL;
}

return E_SUCCESS;
}

```

```

2075
2076 * Is There prev Backup For Time
2077 *
2078 * Function Description:
2079 * Determine if a backup exists prior to the specified time
2080 *
2081 * Parameters:
2082 * context (I) - Pointer to the restore context
2083 * thisTime (I) - Time for the query
2084 * flags (I) - Backup constraint flags: e.g., full-only/partial-ok
2085 * isThere (O) - TRUE/FALSE that requested backup does exist
2086 *
2087 * Return Codes:
2088 * E_SUCCESS - operation completed successfully
2089 * EP_RB_RECOVER_XXX - when errors occur accessing catalogs
2090 *
2091
2092
2093 errno_ty RSTPL_IsTherePrevBackupForTime( restore_context *context,
2094 const time_t thisTime,
2095 RSTRPC_backup_flags_ty flags,
2096 boolean_ty *isThere )
2097
2098 1 {
2099     *isThere=FALSE;
2100
2101     struct restoreContextData *rcd=(struct restoreContextData *)(
2102         context->appData);
2103
2104     WiSetNode *wilist=rccd->currentWiSetNode;
2105
2106     if(!wilist)
2107     {
2108         return EP_RB_RECOVER_RC_APP_DATA_NULL;
2109     }
2110
2111     BackupNode *bnode=rccd->currentBackupNode;
2112
2113     if(!bnode)
2114     {
2115         return EP_RB_RECOVER_RC_APP_DATA_NULL;
2116     }
2117
2118
2119     /*
2120     * Loop over all backup objects in this set looking for one which is
2121     * previous to this one.
2122     */
2123     for(RestoreNode *rn=wilist->getFirstChild();rn;
2124         rn=wilist->getNextChild())
2125     {
2126         if((flags==BACKUP_SELECTION_FLAG_PARTIAL_OK) ||
2127             (BackupNode *bn=(BackupNode *)rn;
2128             RnPropertyDate *dp=(RnPropertyDate *)rn->getProperty(
2129             PROPERTY_BACKUP_DATE));
2130             time_t btime=dp->getValue()->unixTime();
2131
2132         if(thisTime > btime)
2133         {
2134             *isThere=TRUE;
2135         }
2136     }
2137 }
2138
2139 1 return E_SUCCESS;
2140

```

```

2143
2144 * Is There NextBackup For Time
2145
2146 * Function Description:
2147 * Determine if a backup exists after to the specified time
2148 *
2149 * Parameters:
2150 *   context (I) - Pointer to the restore context
2151 *   thistime(I) - Time for the query
2152 *   flags (
2153 *     I) - Backup constraint flags: e.g., full-only/partial-ok
2154 *   isThere (O) - TRUE/FALSE that requested backup does exist
2155 *
2156 * Return Codes:
2157 *   E_SUCCESS - operation completed successfully
2158 *   E_P_RB_RECOVER_xxx - when errors occur accessing catalogs
* ****

```

```

2211  ****
2212  * Identify:
2213  *
2214  * This function is called once, to identify and validate the plug-in with
2215  * regard to the operating restore engine. The version number is checked
2216  * for compatibility with the restore engine, and the optional features
2217  * of the plug-in are specified.
2218  *
2219  * Parameters:
2220  * pi_defs (
2221  *   0) - address of the structure identifying the plugin to the
2222  *   restore engine. Its fields consist of:
2223  *   name - Name of the backup application (64 byte buffer address)
2224  *   options - Bit mask identifying the optional plug-in features
2225  *   The bit definitions for this parameter ( RSTPI_OPTION_MASK...) are
2226  *   defined above.
2227  *   wi_types - pointer to array of workitem types supported by the
2228  *   num_types - number of witypes in the witypes array
2229  *
2230  * Returns:
2231  *   E_SUCCESS      on success
2232  *   EP_RB_RECOVER_XXX on error
2233  *
2234  ****
2235  errno_ty RSTPI_Identifier( const struct pluginIData **pi_defs )
2236  {
2237  static char types[2] = {WI_TYPE_DC_NETWORK,
2238  1
2239  2
2240  2
2241  2
2242  2
2243  2
2244  2
2245  1
2246  1
2247  1
2248  1
2249  1
2250  1
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3398
3399
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3498
3499
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3598
3599
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3698
3699
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3798
3799
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3898
3899
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3998
3999
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4098
4099
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4159
4160
4161
4162
4163
4164
4165
4166
4167
4168
4169
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4198
4199
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4239
4240
4241
4242

```

```

2280 / ****
2281 * Does Alternate Exist
2282 *
2283 * This routine determines if an alternate trailset exists for the
2284 * given template.
2285 *
2286 * Parameters:
2287 *   context (I) - Pointer to the restore context
2288 *   templateName (I) - The name of the template to look for
2289 *   exists (I)
2290 *   O) - Return flag for whether or not the alternate exists
2291 * ****
2292 *
2293 *errno_ty
2294 RSPI_DoesAlternateExist( restore_context
2295 *context,
2296 const template_name_ty templateName,
2297 boolean_ty
2298 1 { * **** TEMPORARY,
2299 1 * until we get the real version: ****
2300 1 *exists = FALSE;
2301 1 return E_SUCCESS;
}

```

Page 91 of 92
restore_dcp/plugin.cc 75
Thu Jan 03 12:52:58 2008

Page 92 of 92
restore_dcp/plugin.cc 76
Thu Jan 03 12:52:58 2008